CrossTalk

March 2007 The Journal of Defense Software Engineering Vol. 20 No. 3

Software Security

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **MAR 2007**	2. REPORT TYPE	3. DATES COVERED **00-00-2007 to 00-00-2007**

4. TITLE AND SUBTITLE **CrossTalk: The Journal of Defense Software Engineering. Volume 20, Number 3, March 2007**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **OO-ALC/MASE,6022 Fir Ave,Hill AFB,UT,84056-5820**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**32**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

Software Security

On The Cover
Cover Design by Kent Bingham

Additional art services provided by Janna Jensen

Departments

CrossTalk

Co-Sponsors:	
DoD-CIO	*The Honorable John Grimes*
NAVAIR	*Jeff Schwalb*
76 SMXG	*Kevin Stamey*
309 SMXG	*Randy Hill*
402 SMXG	*Diane Suchan*
DHS	*Joe Jarzombek*
Staff:	
Managing Director	*Brent Baxter*
Publisher	*Elizabeth Starrett*
Managing Editor	*Kase Johnstun*
Associate Editor	*Chelene Fortier-Lozancich*
Article Coordinator	*Nicole Kentta*
Phone	(801) 775-5555
E-mail	crosstalk.staff@hill.af.mil
CrossTalk Online	www.stsc.hill.af.mil/crosstalk

CrossTalk, The Journal of Defense Software Engineering is co-sponsored by the Department of Defense Chief Information Office (DoD-CIO); U.S. Navy (USN); U.S. Air Force (USAF); and the U.S. Department of Homeland Security (DHS). DoD-CIO co-sponsor: Assistant Secretary of Defense (Networks and Information Integration). USN co-sponsor: Naval Air Systems Command. USAF co-sponsors: Oklahoma City-Air Logistics Center (ALC) 76 Software Maintenance Group (SMXG); Ogden-ALC 309 SMXG; and Warner Robins-ALC 402 SMXG. DHS co-sponsor: National Cyber Security Division of the Office of Infrastructure Protection.

The USAF Software Technology Support Center (STSC) is the publisher of CrossTalk, providing both editorial oversight and technical review of the journal. CrossTalk's mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

S T S C

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail us or use the form on p. 21.

517 SMXS/MXDEA
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CrossTalk editorial board prior to publication. Please follow the Author Guidelines, available at <www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf>. CrossTalk does not pay for submissions. Articles published in CrossTalk remain the property of the authors and may be submitted to other publications.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with CrossTalk.

Trademarks and Endorsements: This Department of Defense (DoD) journal is an authorized publication for members of the DoD. Contents of CrossTalk are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the co-sponsors, or the STSC. All product names referenced in this issue are trademarks of their companies.

CrossTalk Online Services: See <www.stsc.hill.af.mil/crosstalk>, call (801) 777-0857 or e-mail <stsc.webmaster@hill.af.mil>.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

Collaborating for Secure Software

I have worked with Joe Jarzombek and the Department of Homeland Security (DHS) over the past years and have seen the result of their commitment to software security in a growing number of resources for assistance. While most people have a basic definition of security, I'll share the DHS definition: *Protection against intentional subversion or forced failure. Preservation of the three subordinate properties that make up security – availability, integrity, and confidentiality.*

With all the current efforts to secure software, our nation still encounters attacks. It is estimated that 90 percent of reported security incidents result from exploits against defects in the software design or code. This is especially troubling because many of the vulnerabilities that enable these attacks can be prevented with the use of solid software engineering principles. Given that, why do we still have difficulty ensuring the integrity of software that is so key to protecting the infrastructure from threats and vulnerabilities, reducing overall risk to cyber attack?

In a continuing effort to develop defenses to software attacks and educate software personnel on implementing these defenses, DHS is involved with multiple initiatives. Their primary goals while deploying software security practices include the following:

- Exploitable faults and other weaknesses are avoided by well-intentioned developers.
- The likelihood is greatly reduced or eliminated that malicious developers can intentionally implant exploitable faults and weaknesses or malicious logic into the software.
- The software is attack-resistant, attack-tolerant, and attack-resilient.

To this end, a great deal of attention has been focused on their BuildSecurityIn initiative. By accessing <http://BuildSecurityIn.us-cert.gov>, developers, acquirers, and users can find a broad range of information about best practices, tools, guidelines, rules, principles, and other knowledge to help organizations build secure and reliable software. Through DHS' sponsorship of conferences and workshops, a common body of knowledge and a repository of practical guidance for software developers and architects are being produced to improve the quality, reliability, and dependability of software. In collaboration with industry, academia, and government partners, DHS' approach to addressing software assurance encompasses the following components:

- People – Education and training for developers and users.
- Processes – Practical guidelines and best practices for the development of secure software.
- Technology – Tools for evaluating software vulnerabilities and quality.
- Acquisition – Specifications and guidelines for acquisition and outsourcing.

I am pleased that CrossTalk continues to be a forum for educating the software community on software security. Some of this month's authors have personally worked with DHS on the topics discussed. We begin with Robert A. Martin's discussion of the Common Weaknesses dictionary in *Being Explicit About Security Weaknesses*. Standards provide consolidated resources for the software community, especially developers, to glean specific techniques for developing secure software. We build on this with *Secure Coding Standards* by James W. Moore and Robert C. Seacord and *How a Variety of Information Assurance Methods Delivers Software Security in the United Kingdom* by Kevin Sloan and Mike Ormerod. Going beyond standards, we begin our discussion of actual practices with Idongesit Mkpong-Ruffin and Dr. David A. Umphress' article, *High-Leverage Techniques for Software Security*. Next, *Baking in Security During the Systems Development Life Cycle* by Kwok H. Cheng emphasizes the DHS philosophy that security must be part of the entire software development process. We conclude with Mel Crocker's insights on certifications and technological advances to enable information sharing across platforms in *Cross-Domain Information Sharing in a Tactical Environment*.

I echo Mr. Jarzombek's challenges in past issues to our readers to take the time to understand and apply these principles and techniques. Through this effort, we can continue to expand the software security community of practice.

Elizabeth Starrett

Elizabeth Starrett
Publisher

Being Explicit About Security Weaknesses

Robert A. Martin
MITRE Corporation

Software acquirers want assurance that the products they are obtaining are reviewed for known types of security weaknesses. Acquisition groups in large government and private organizations are beginning to use such reviews as part of future contracts, but the tools and services for performing them are new and, until recently, there was no nomenclature, taxonomy, or standard to define their capabilities and coverage. A standard dictionary of software security weaknesses has been created by the community to serve as a unifying language of discourse and as a measuring stick for comparing tools and services.

More and more organizations want assurance that the software products they acquire and develop are free of known types of security weaknesses. High-quality tools and services for finding security weaknesses in code are new. The question of which tool/service is appropriate/better for a particular job is hard to answer given the lack of structure and definition in the software product assessment industry.

There are several ongoing efforts to begin to resolve some of these shortcomings, including the Department of Homeland Security (DHS) National Cyber Security Division (NCSD)-sponsored Software Assurance Metrics and Tool Evaluation (SAMATE) project [1] led by the National Institute of Standards and Technology (NIST), the Object Management Group (OMG) Software Assurance (SwA) Special Interest Group (SIG) [2], and the Department of Defense (DoD)-sponsored Code Assessment Methodology Project, which is part of the Protection of Vital Data effort [3] conducted by Concurrent Technologies Corporation, among others. While these efforts are well placed, timely in their objectives, and will surely yield high value in the end, they require a common description of the underlying security weaknesses that can lead to exploitable vulnerabilities in the software that they are targeted to resolve. Without such a common description, these efforts, as well as the DoD's own software and systems assurance efforts, cannot move forward in a meaningful fashion or be aligned and integrated with each other to provide the needed answers to secure our networked systems.

A Different Approach

Past attempts at developing this kind of effort have been limited by a very narrow technical domain focus or have largely focused on high-level theories, taxonomies, or schemes that do not reach the level of detail or variety of security issues that are found in today's products. As an alternate approach, under sponsorship of DHS NCSD, and as part of MITRE's participation in the DHS-sponsored NIST SAMATE effort, MITRE investigated the possibility of leveraging the Common Vulnerabilities and Exposures (CVE) initiative's experience in analyzing more than 20,000 real-world vulnerabilities reported and discussed by industry and academia.

As part of the creation of the CVE list [4] that is used as the source of vulnerabilities for the National Vulnerability Database [5], MITRE's CVE initiative during the last six years has developed a preliminary classification and categorization of vulnerabilities, attacks, faults, and other concepts that can be used to help define this arena. However, the original groupings used in the development of CVE, while sufficient for that task, were too rough to be used to identify and categorize the functionality found within the offerings of the code security assessment industry. For example, in order to support the development of CVE content, it is sufficient to separate the reported vulnerabilities in products into working categories such as weak/bad authentication, buffer overflow, cryptographic error, denial of service, directory traversal, information leak, or cross-site scripting. For assessing code, however, this granularity of classification groupings was too large and indefinite. Of the categories listed, for example, cross-site scripting actually has eight different types of issues that need to be addressed or looked for when assessing code; buffer overflow covered 10 different code constructs to look for.

So, to support use in code assessment, additional fidelity and succinctness was needed as well as additional details and descriptive information for each of the different categories such as effects, behaviors, and the implementation details. The preliminary classification and categorization work used in the development of CVE was revised to address the types of issues discussed above and the result was called the Preliminary List of Vulnerability Examples for Researchers (PLOVER) [6]. PLOVER was a document that listed more than 1,500 diverse, real-world examples of vulnerabilities, identified by their CVE name. The vulnerabilities are organized within a detailed conceptual framework that enumerates the 300 individual types of weaknesses that cause the vulnerabilities. The weaknesses were simply grouped within 28 higher-level categories with a large number of real-world vulnerability examples for each type of weakness. PLOVER represents the first cut of a truly bottom-up effort to take real-world observed, exploitable vulnerabilities that *do* exist in code, to abstract them and group them into common classes representing more general potential weaknesses that *could* lead to exploitable vulnerabilities in code, and then, finally to organize them in an appropriate relative structure so as to make them accessible and useful to a diverse set of audiences for a diverse set of purposes.

Creating a Community Effort

As part of the DoD/DHS SwA working groups and the NIST SAMATE project, MITRE fostered the creation of a community of partners from industry, academia, and government to develop, review, use, and support a common weaknesses dictionary that can be used by those looking for weaknesses in code, design, or architecture as well as those teaching and training software developers about the code, design, or architecture weaknesses that they should avoid due to the security problems they can have on applications, systems, and networks. The effort is called the Common Weakness Enumeration (CWE) initiative. The work from PLOVER became the major source of content for draft one of the CWE dictionary.

An important element of the CWE initiative is to be transparent to all on what we are doing, how we are doing it, and what we are using to develop the CWE

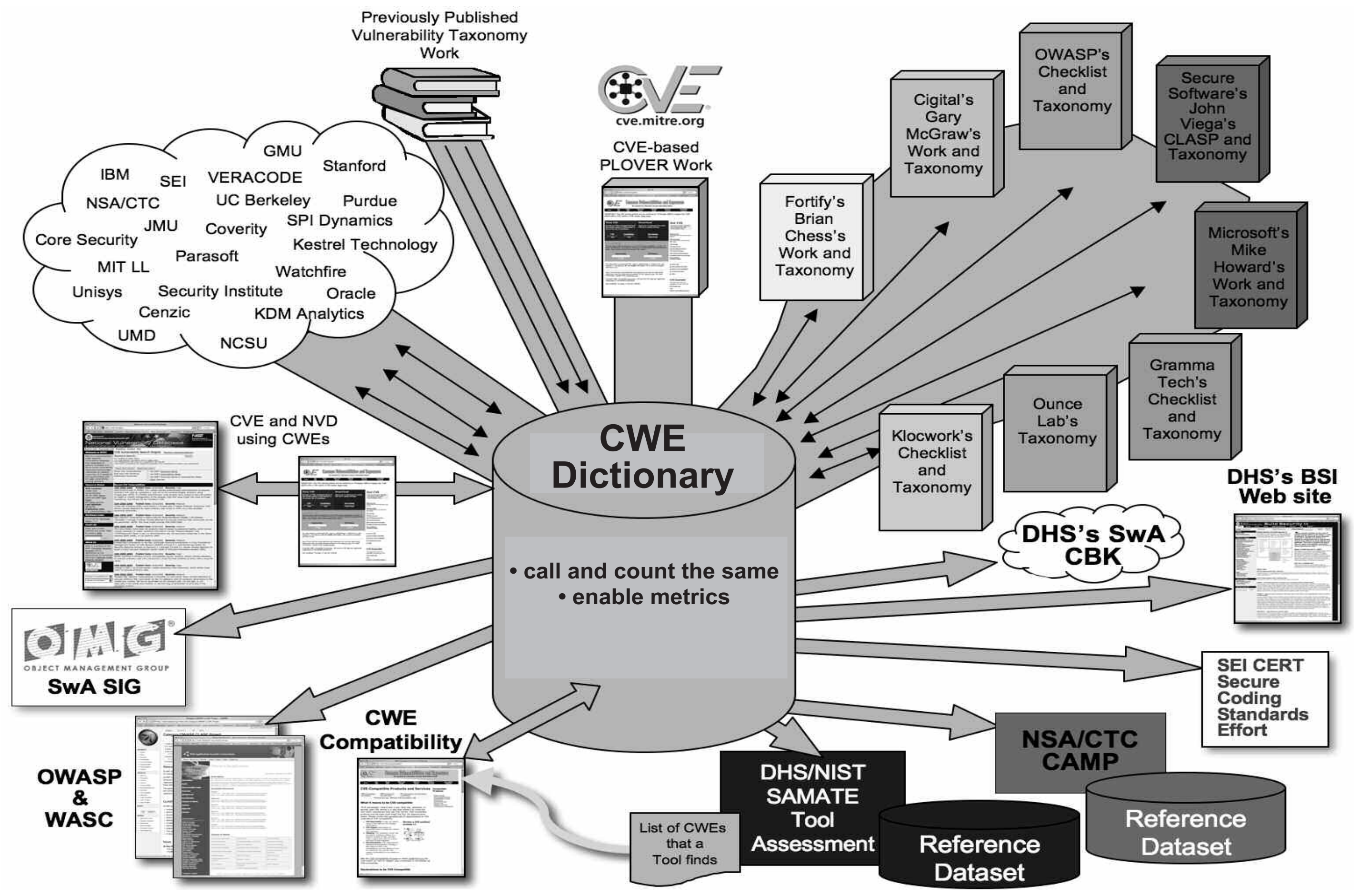

Previously Published Vulnerability Taxonomy Work
CVE
cve.mitre.org
CVE-based PLOVER Work
IBM
SEI
VERACODE
GMU
Stanford
NSA/CTC
UC Berkeley
Purdue
JMU
Coverity
SPI Dynamics
Core Security
Kestrel Technology
Parasoft
MIT LL
Watchfire
Unisys
Security Institute
Oracle
Cenzic
KDM Analytics
UMD
NCSU
Fortify's Brian Chess's Work and Taxonomy
Cigital's Gary McGraw's Work and Taxonomy
OWASP's Checklist and Taxonomy
Secure Software's John Viega's CLASP and Taxonomy
Microsoft's Mike Howard's Work and Taxonomy
Gramma Tech's Checklist and Taxonomy
Ounce Lab's Taxonomy
Klocwork's Checklist and Taxonomy
CVE and NVD using CWEs
CWE Dictionary
• call and count the same
• enable metrics
DHS's BSI Web site
DHS's SwA CBK
OMG
OBJECT MANAGEMENT GROUP
SwA SIG
SEI CERT Secure Coding Standards Effort
CWE Compatibility
OWASP & WASC
NSA/CTC CAMP
DHS/NIST SAMATE Tool Assessment
List of CWEs that a Tool finds
Reference Dataset
Reference Dataset

Figure 1: *The CWE Effort's Context and Community*

dictionary. We believe this transparency is important during the initial creation of the CWE dictionary so that all of the participants in the CWE community are comfortable with the end result and will not be hesitant about incorporating CWE into what they do. Figure 1 shows the overall CWE context and community involvement of the effort. We believe the transparency should also be available to participants and users that will visit after the initial CWE dictionary is available on the CWE Web site [7]; all of the publicly available source content is being hosted on the site for anyone to review or use for their own research and analysis.

Currently, more than 41 organizations, shown in Table 1 (see page 6), are participating in the creation and population of the CWE dictionary.

Kick-Starting a Dictionary

To continue the creation of the CWE dictionary, we brought together as much public content as possible using the following three primary sources:

- The PLOVER collection [6] that identified more than 300 weakness types created by determining the root issues behind 1,500 of the vulnerabilities in the CVE List [4].
- The Comprehensive, Lightweight Application Security Process (CLASP) from Secure Software, which yielded more than 90 weakness concepts [8].
- The issues contained in Fortify's *Seven Pernicious Kingdoms* papers, which contributed more than 110 weakness concepts [9].

Working from these collections as well as those contained in the 13 other publicly available information sources listed on the CWE Web site *sources* page, we developed the first draft of the CWE list, which entailed almost 500 separate weaknesses. It took approximately six months to move from what we created in PLOVER to the first draft of CWE. The CWE content is captured in an XML document and follows the CWE schema. Two months later, we updated CWE to draft 2 with the incorporation of changes that included cleaning up the names of items, reworking the structure, and filling in the descriptive details for many more of the items. The first change to the CWE schema came about with the addition of language and platform ties for weaknesses and the addition of specific CWE identifications for each weakness.

Covering What Tools Find

While the third draft of CWE continued expanding the descriptions and improving the consistency and linkages, subsequent drafts will incorporate the specific details and descriptions of the 16 organizations that have agreed to contribute their intellectual property to the CWE initiative. Under non-disclosure agreements with MITRE, which allow the merged collection of their individual contributions to be publicly shared in the CWE List, Application Security Consortium, Cenzec, Core Security, Coverity, Fortify, Interoperability Clearinghouse, Klocwork, Ounce Labs, Parasoft, proServices Corporation, Secure Software, Security Innovation Inc., SofCheck, SPI Dynamics, Veracode, and Watchfire are all contributing their knowledge and experience to building out the CWE dictionary. Draft 4 is the first draft version to include details from this set of information sources.

Draft 5 of CWE encompasses more than 600 nodes with specific details and examples of weaknesses for many of the entries. Figure 2 shows the transition from PLOVER to CWE drafts 1-5 and the content structure changes that occurred during the revisions. While the initial transi-

- AppSIC, LLC.
- Aspect Security
- Cenzic, Inc.
- Center for Education and Research in Information Assurance and Security/ Purdue University
- Computer Emergency Response Team/ Coordination Center (CERT/CC)
- Cigital, Inc.
- Code Scan Labs
- Core Security Technologies
- Coverity, Inc.
- Fortify Software, Inc.
- IBM
- Interoperability Clearing House
- James Madison University
- Johns Hopkins University Applied Physics Laboratory
- KDM Analytics
- Kestrel Technology
- Klocwork, Inc.
- Microsoft Corporation
- Massachusets Institute of Technology Lincoln Labs
- MITRE Corporation
- NIST
- National Security Agency (NSA)
- North Carolina State University
- OMG
- Open Web Application Security Project (OWASP)
- Oracle Corporation
- Ounce Labs, Inc.
- Palamida
- Parasoft Corporation
- proServices Corporation
- Secure Software, Inc.
- Security Innovation, Inc.
- Security University
- Semantic Designs, Inc.
- SofCheck, Inc.
- SPI Dynamics, Inc.
- Unisys
- VERACODE
- Watchfire Corporation
- Web Application Security Consortium (WASC)
- Whitehat Security, Inc.

Table 1: *The CWE Community*

tion from PLOVER to CWE took six months, each subsequent updated draft has occurred on a bimonthly basis.

In addition to the sources supplying specific knowledge from tools or analysts, we are also leveraging the work, ideas, and contributions of researchers at Carnegie Mellon's CERT/CC, IBM, KDM Analytics, Kestrel Technology, MIT's Lincoln Labs, North Carolina State University, Oracle, the OWASP, Security Institute, Unisys, the WASC, Whitehat Security, and any other interested parties that wish to contribute.

The merging and combining of the contributed materials is being incorporated into several of drafts of CWE (draft 6 in February, 2007 and draft 7 in May, 2007), which will be available for open community comments and refinement as CWE moves forward. A major part of the future work will be refining and defining the required attributes of CWE elements into a more formal schema defining the metadata structure necessary to support the various uses of CWE dictionary. Figure 3 shows a sample of the descriptive content of an entry from CWE draft 5. This example is for the Double Free weakness, CWE identification (ID) 415.

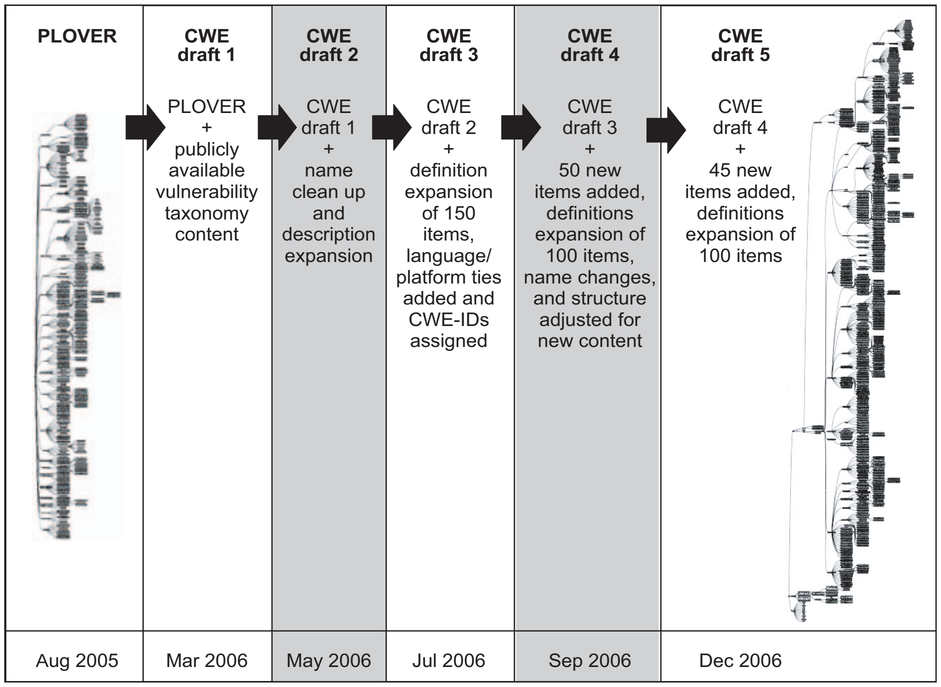
However, the CWE schema will also be driven by the need to align with and support the SAMATE and OMG SwA SIG efforts that are developing software metrics, software security tool metrics, the software security tool survey, the methodology for validating software security tool claims, and reference datasets for testing.

For example, a major aspect of the SAMATE project is the development and open sharing of test applications that have been salted with known weaknesses so that those wishing to see how effective a particular tool or technique is in finding that type of weakness will have test materials readily available. These test sets are referred to as the SAMATE test reference datasets (TRDs). NIST has chosen to organize the SAMATE TRDs by CWE weakness type and will also include varying levels of complexity, as appropriate to each type of weakness, so that tools that are more or less effective in finding complex examples of a particular CWE weakness can be identified. Correct constructs that are closely aligned to the CWEs but are correct implementations will also be included in the TRDs to help identify the false-positive effectiveness of the tools. Adding complexity descriptions to the CWE schema will allow SAMATE and CWE to continue to support each other.

The OMG's SwA SIG, which is using CWEs as one type of software issue that tools will need to be able to locate within the eventual OMG SwA technology approach, needs more formal descriptions of the weaknesses in CWE to allow their technological approaches to apply. OMG's planned approach for this is the use of their Semantics of Business Vocabulary and Rules (SBVR) language to articulate formal language expressions of the different CWEs. The CWE schema will have to be enhanced to allow SBVR expressions of each CWE to be included. The CWE will house the official version of the SBVR expression of that CWE.

The CWE dictionary content is already provided in several formats and will have additional formats and views added into its contents as the initiative proceeds. Currently one of the ways for viewing CWE is through the CWE content page that contains an expanding/ contracting hierarchical *taxonometric* view while another is through an alphabetic dictionary. The end items in the hierarchical view are hyperlinked to their respective dictionary entries. Graphical depictions of CWE content, as well as the contributing sources, are also available. Finally, the XML and XML Schema Definition (XSD) for CWE are provided for those who wish

Figure 2: *From PLOVER to CWE, Drafts 1-5*


PLOVER
CWE draft 1
CWE draft 2
CWE draft 3
CWE draft 4
CWE draft 5
PLOVER + publicly available vulnerability taxonomy content
CWE draft 1 + name clean up and description expansion
CWE draft 2 + definition expansion of 150 items, language/ platform ties added and CWE-IDs assigned
CWE draft 3 + 50 new items added, definitions expansion of 100 items, name changes, and structure adjusted for new content
CWE draft 4 + 45 new items added, definitions expansion of 100 items
Aug 2005
Mar 2006
May 2006
Jul 2006
Sep 2006
Dec 2006

® CERT is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

to do their own analysis/review with other tools. Dot notation representations, a standard method for encoding graphical plots of information, will be added in the future.

Finally, a process to acknowledge capabilities that incorporate CWEs has been established. This CWE Compatibility and CWE Effectiveness program is similar to the certification and branding program used by the CVE effort but has two distinct parts, compatibility and effectiveness. The basic stages of the compatibility program are a formalized process for capability owners to publicly declare their use of CWEs and a public documentation of how their capability fulfills the requirements for finding those CWEs. The effectiveness program, which only applies to assessment capabilities, consists of a public declaration about which CWEs a capability covers and collection of publicly available test results, showing how effective the capability is in finding those CWEs.

Additional Impact and Transition Opportunities Tied to CWE

The establishment of the CWE effort is yielding consequences of the following three types: direct impact and value, alignment with and support of other existing efforts, and enablement of new follow-on efforts to provide value that is not currently being pursued.

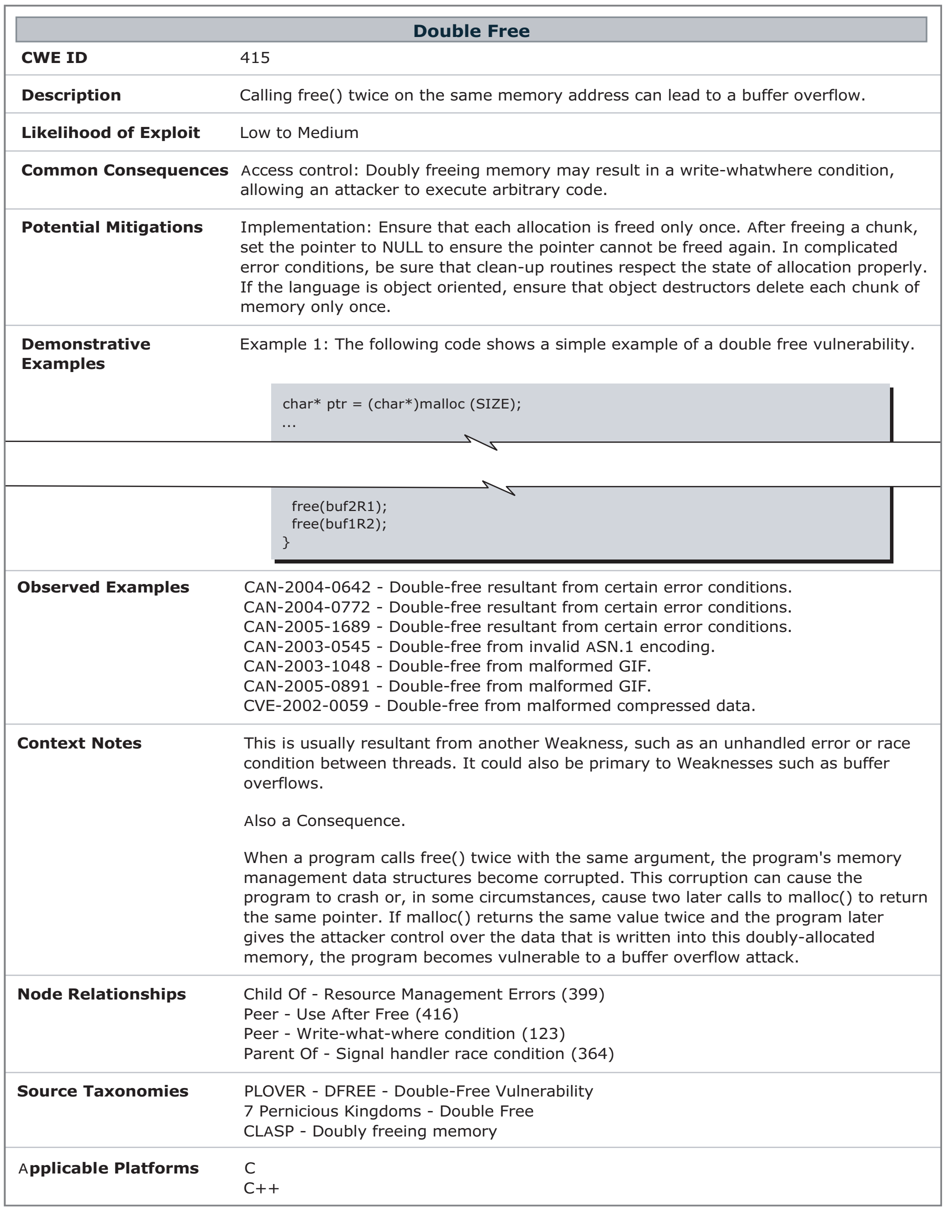
The direct impacts include the following:

- Providing a common language of discourse for discussing, finding, and dealing with the causes of software security vulnerabilities as they are manifested in code, design, or architecture.
- Allowing purchasers to compare, evaluate, and select software security tools and services that are most appropriate to their needs – including having some level of assurance of the assortment of CWEs that a given tool would find. Software purchasers will be able to compare coverage of tool and service offerings against the list of CWEs and the programming languages that are used in the software they are acquiring.
- Enabling the verification of coverage claims made by software security tool vendors and service providers (this is supported through CWE metadata and alignment with the SAMATE reference dataset).
- Enabling government and industry to leverage this standardization in their acquisition contractual terms and conditions.

There will also be a variety of alignment opportunities where other security-related efforts and CWE can leverage each other to the benefit of both. Examples of the synergies that are possible include the following:

- Mapping of CWEs to CVEs that would help bridge the gap between the potential sources of vulnerabilities and examples of their observed instances providing concrete information for better understanding the CWEs and providing some validation of the CWEs themselves.
- Creating a validation framework for tool/service vendor claims, whether used by the purchasers themselves or through a third-party validation service, would be able to heavily leverage the common weaknesses dictionary as its basis of analysis. To support this, the community would need to define the mechanisms used to exploit the various CWEs for the purposes of helping to clarify the CWE groupings and come up with verification methods for validating the effectiveness of tools to identify the presence of CWEs in code. The effectiveness of these test approaches could be explored with the goal of identifying a method or methods that are effective and economical to apply to the validation process.
- Establishing a bi-directional alignment between the common weaknesses enumeration and the SAMATE metrics effort.
- Using the SAMATE software security tool and services survey effort to leverage this common weaknesses dictionary as part of the capability framework to effectively and unambiguously describe various tools and services in a consistent apples-to-apples fashion.
- Mapping between the CWEs and the common attack pattern enumeration and characterization effort that would

Figure 3: *Entry for CWE-ID 415, Double Free Weakness*



Double Free	
CWE ID	415
Description	Calling free() twice on the same memory address can lead to a buffer overflow.
Likelihood of Exploit	Low to Medium
Common Consequences	Access control: Doubly freeing memory may result in a write-whatwhere condition, allowing an attacker to execute arbitrary code.
Potential Mitigations	Implementation: Ensure that each allocation is freed only once. After freeing a chunk, set the pointer to NULL to ensure the pointer cannot be freed again. In complicated error conditions, be sure that clean-up routines respect the state of allocation properly. If the language is object oriented, ensure that object destructors delete each chunk of memory only once.
Demonstrative Examples	Example 1: The following code shows a simple example of a double free vulnerability. `char* ptr = (char*)malloc (SIZE);` `...` `free(buf2R1);` `free(buf1R2);` `}`
Observed Examples	CAN-2004-0642 - Double-free resultant from certain error conditions. CAN-2004-0772 - Double-free resultant from certain error conditions. CAN-2005-1689 - Double-free resultant from certain error conditions. CAN-2003-0545 - Double-free from invalid ASN.1 encoding. CAN-2003-1048 - Double-free from malformed GIF. CAN-2005-0891 - Double-free from malformed GIF. CVE-2002-0059 - Double-free from malformed compressed data.
Context Notes	This is usually resultant from another Weakness, such as an unhandled error or race condition between threads. It could also be primary to Weaknesses such as buffer overflows. Also a Consequence. When a program calls free() twice with the same argument, the program's memory management data structures become corrupted. This corruption can cause the program to crash or, in some circumstances, cause two later calls to malloc() to return the same pointer. If malloc() returns the same value twice and the program later gives the attacker control over the data that is written into this doubly-allocated memory, the program becomes vulnerable to a buffer overflow attack.
Node Relationships	Child Of - Resource Management Errors (399) Peer - Use After Free (416) Peer - Write-what-where condition (123) Parent Of - Signal handler race condition (364)
Source Taxonomies	PLOVER - DFREE - Double-Free Vulnerability 7 Pernicious Kingdoms - Double Free CLASP - Doubly freeing memory
Applicable Platforms	C C++

provide the users of these resources the ability to quickly identify the particular weaknesses that are targeted by various types of attacks and to better understand the context of individual weaknesses through understanding how they would typically be targeted for exploitation. In combination, these two resources offer significantly higher value than either does on its own.
- Bi-directional mapping between CWEs and coding rules, such as those deployed as part of the DHS NCSD BuildSecurityIn Web site [10], would be used by tools and in manual code inspections to identify common weaknesses in software.
- Incorporating CWE into the DHS NCSD SwA common body of knowledge, hosted on the BuildSecurityIn Web site.
- Leveraging of the OMG technologies to articulate formal, machine-parsable definitions of the CWEs to support analysis of applications within the OMG standards-based tools and models.

Finally, there are two follow-on opportunities that are currently not being pursued but could provide significant added value to the software security industry:
- Expansion of the coding rules catalog on the DHS BuildSecurityIn Web site to include full mapping against the CWEs for all relevant technical domains.
- Identification and definition of specific domains (language, platform, functionality, etc.) and relevant protection profiles based on coverage of CWEs. These domains and profiles could provide a valuable tool to security testing strategy and planning efforts.

Conclusion

This work is already helping to shape and mature the code security assessment industry, and it promises to dramatically accelerate the use and utility of automation-based assessment capabilities for organizations and the software systems they acquire, develop, and use.◆

Acknowledgments

The work contained in this paper was funded by DHS NCSD and is based on the efforts of a large number of individuals, but special thanks is made for the contributions of Steve Christey, Janis Kenderdine, Conor Harris, David Harris, and Sean Barnum.

References

1. NIST. "The Software Assurance Metrics and Tool Evaluation (SAMATE) Project." Jan. 2007 <http://samate.nist.gov>.
2. OMG. Object Management Group. Jan. 2007 <http://swa.omg.org>.
3. Concurrent Technologies Corporation. The Code Assessment Methodology Project. Jan. 2007 <www.ctc.com>.
4. MITRE Corporation. The Common Vulnerabilities and Exposures (CVE) Initiative. Jan. 2007 <http://cve.mitre.org>.
5. NIST. National Vulnerability Database. Jan. 2007 <http://nvd.nist.gov>.
6. MITRE Corporation. The Preliminary List of Vulnerability Examples for Researchers. Dec. 2006 <http://cve.mitre.org/docs/plover/>.
7. MITRE Corporation. The Common Weakness Enumeration Initiative <http://cwe.mitre.org>.
8. Viega, J. The CLASP Application Security Process. Secure Software, Inc., 2005 <www.securesoftware.com>.
9. McGraw, G., B. Chess, and K. Tsipenyuk. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors." NIST Workshop on Software Security Assurance Tools, Techniques, and Metrics, Long Beach, CA, Nov. 2005.
10. DHS NCSD. BuildSecurityIn. Dec. 2006 <http://buildsecurityin.us-cert.gov>.

About the Author

Robert A. Martin is a principal engineer in MITRE's Information and Computing Technologies Division. For the past seven years, his efforts have been focused on the interplay of risk management, cyber security standards, critical infrastructure protection, and the use of software-based technologies and services. Martin is a member of the Association for Computing Machinery, Armed Forces Communications and Electronics Association, Institute of Electrical and Electronics Engineers (IEEE), and the IEEE Computer Society. He has a bachelor's degree and a master's degree in electrical engineering from Rensselaer Polytechnic Institute, and a master's of business degree from Babson College.

MITRE Corporation
202 Burlington RD
Bedford, MA 01730-1420
Phone: (781) 271-3001
Fax: (781) 271-8500
E-mail: ramartin@mitre.org

Secure Coding Standards

James W. Moore
The MITRE Corporation

Robert C. Seacord
Software Engineering Institute

Inherent weaknesses in programming languages contribute to software vulnerabilities. Increasingly, organizations are producing standards to improve software security. Current efforts to develop software security standards are surveyed, and two such efforts are described in detail. An international standards group is writing a document providing guidance to users of programming languages on how to avoid the vulnerabilities that exist in the programming language selected for a particular project. Carnegie Mellon University's (CMU's) Computer Emergency Response Team (CERT) is developing secure coding practices for the C and C++ programming languages[1].

Today's dependency on networked software systems has been matched by an increase in the number of attacks against governments, corporations, educational institutions, and individuals. These attacks result in the loss and compromise of sensitive data, system damage, lost productivity, and financial loss. To address this growing threat, the introduction of software vulnerabilities during development and ongoing maintenance must be significantly reduced, if not eliminated.

It is no secret that common, everyday software defects cause the majority of software vulnerabilities. Poor development practices cause numerous delivered defects, some of which can lead to vulnerabilities. Software developers repair vulnerabilities as they are reported; cycles of patch and install follow. However, there are so many patches to install that system administrators cannot keep up with the job. Often the attackers analyze the patches for clues to attacking unpatched systems. Sometimes the patches themselves contain security defects. The patch-oriented strategy of responding to security defects is not working. There is need for a prevention and early security defect-removal strategy.

Software Vulnerabilities

One example of a relatively common programming error that can lead to software vulnerabilities involves the use of formatted output functions in C and C++, as well as some other common languages. For example, a malicious user is able to manipulate the string variable named *output* in the following statement:

```
printf(output);
```

The malicious user will be able to exploit this vulnerability to execute code with the permissions of the vulnerable process [1]. This is largely a consequence of the little known %**n** conversion specifier that instructs the formatted output function to write an integer value to a specified address. The error is the passing of untrusted data as a format string. The solution is as easy as providing a static format specification:

```
printf("%s", output);
```

Or, even more succinctly, using the puts() function so that no formatting is necessary or possible, as shown in the following example:

```
puts(output);
```

Format output functions can also lead to vulnerabilities in other languages as is shown in the following examples:

- **Perl:** could alter values, the consequence of which depends on how the value is used.
- **PHP:** log avoidance (fails quietly).
- **Python:** information leak or denial-of-service (DoS) attacks.
- **Ruby:** DoS attacks.

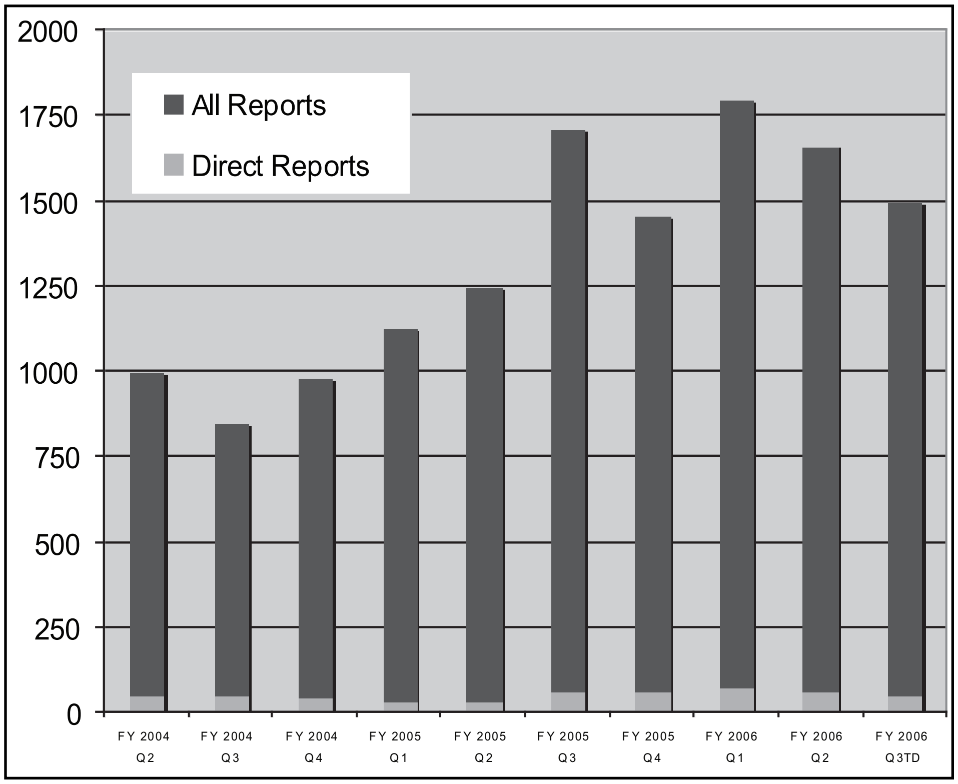
As long as developers are unaware of the security risks, it is likely that common programming errors, such as allowing untrusted data to be incorporated into a format string, will result in software vulnerabilities being operationally deployed.

If root causes of software vulnerabilities are not addressed, software vulnerability reports are likely to continue on the upward trend as shown in Figure 1. Nearly 4,000 vulnerabilities were reported in the first half of 2006 alone.

International Standardization Efforts

Inherent weaknesses in programming languages such as C, C++, Fortran, Ada, and COBOL are a contributing factor to software vulnerabilities. Many of these languages were specified in the days when ubiquitous networking language designers

Figure 1: *Vulnerabilities Reported to CERT/Coordination Center (CC)*



were not greatly concerned with the prospect of attacks by external parties. Many languages were predominately designed for flexibility, ease of use, and performance. As a result, many of the languages, in effect, invite coders to use insecure coding constructs. One result has been the growth of usage guidelines that inform and encourage coders to use secure alternatives to easily attacked constructs. Another result is the development of recent projects by standards bodies to provide secure alternatives.

Increasingly, standards organizations are working on ways to improve software security. Accomplishing change through standards organizations can be harder than accomplishing change at any other organizational level, but when successful, can have a broader impact across the industry. The international standards bodies – International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) – are working on a number of projects that affect software security:

- The ISO Technical Management Board (TMB) performs strategic planning and coordination for ISO. Currently, its advisory group on security is coordinating standards efforts. The ISO TMB working group on risk management is providing overall guidance for risk management.
- ISO/IEC Joint Technical Committee (JTC) 1/Standards Committee (SC) 27 is responsible for computing security. Many of its projects concern methods for protecting deployed software <www.ni.din.de/sixcms/detail.php?id=10172>.
- ISO/IEC JTC 1/SC 7 has the responsibility for systems and software engineering. Its standards provide a baseline for the responsible practice of systems and software engineering, including software assurance <www.jtc1-sc7.org/>.
- ISO/IEC JTC 1/SC 22 has the responsibility for programming languages, including their effect on improved software security <www.open-std.org/jtc1/sc22/>. For example, SC 22/ Working Group (WG) 14, the group that is responsible for the standardization of the C programming language, is developing a technical report on C library functions that incorporate bounds checking to mitigate against buffer overflow vulnerabilities [2].

Figure 2 illustrates the relationship of the relevant standards committees.

Other Working Group Vulnerabilities (OWGV)

All programming languages have constructs that are imperfectly defined, implementation-dependent, or difficult to use correctly. As a result, software programs can execute differently than intended by the writer. In some cases, these vulnerabilities can be exploited by an attacker to compromise the safety, security, and privacy of a system.

A new SC 22 group, the OWGV, is addressing the issue of programming language vulnerabilities. The goal of the OWGV <http://aitc.aitcnet.org/isai/>, as described in the proposal for a new work item, is the following:

> … to prepare comparative guidance spanning a large number of programming languages, so that application developers will be better informed regarding the vulnerabilities inherent to candidate languages and the costs of avoiding such vulnerabilities. An additional benefit is that developers will be better prepared to select tooling to assist in the evaluation and avoidance of vulnerabilities. [3]

ISO/IEC JTC 1/SC 22/OWGV is writing a document containing guidance to users of programming languages on how to avoid the vulnerabilities that exist in the programming language selected for a particular project. Implicitly, the guidance may also be helpful in selecting a language for a particular project. Finally, the report may be helpful in choosing tools to assist in evaluating and avoiding vulnerabilities. The tentative schedule calls for publishing this document in January of 2009.

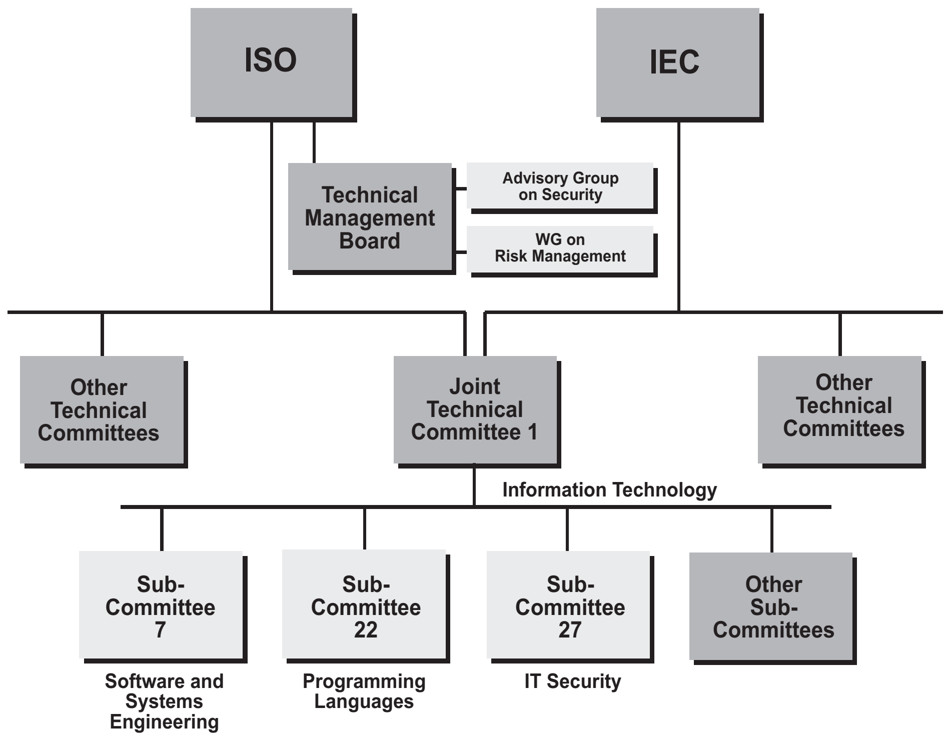
Although work has just begun, it is anticipated that the group's technical report will provide comparative guidance covering a wide variety of languages. Readers would be able to study the vulnerabilities of particular languages they already know well; in addition, readers would be able to apply their existing knowledge of vulnerabilities in the context of programming languages that are unfamiliar.

Currently, the group enjoys the participation of representatives from many of the important programming languages and hopes to attract more. The group plans to obtain information about vulnerabilities and their treatment from initiatives like the common vulnerabilities and exposures database <http://cve.mitre.org> and the CERT secure coding initiative.

CERT Secure Coding Initiative

Additional efforts in developing secure coding standards are originating outside of formal standards organizations. CMU's Software Engineering Institute (SEI) CERT program has deployed a secure coding Web site at <www.securecoding.cert.org> to cooperate with the software development community in codifying a

Figure 2: *Relevant International Standard Committees*


ISO
IEC
Technical Management Board
Advisory Group on Security
WG on Risk Management
Other Technical Committees
Joint Technical Committee 1
Other Technical Committees
Information Technology
Sub-Committee 7
Sub-Committee 22
Sub-Committee 27
Other Sub-Committees
Software and Systems Engineering
Programming Languages
IT Security

practical and effective set of secure coding practices for popular programming languages. These coding practices can be used by software developers to eliminate vulnerabilities before software is operationally deployed.

CERT's initial efforts are focused on the development of secure coding practices for the C and C++ programming languages. C and C++ were selected because a large percentage of critical infrastructure is developed and maintained using these programming languages. C and C++ are popular and viable languages, although they have characteristics that make them prone to security flaws. The CERT C programming language secure coding standard is scheduled for publication in January 2008 while the C++ standard is not scheduled for publication until January 2009. However, working drafts for both documents are available on the secure coding Web site.

There are numerous available sources, both online and in print, containing coding guidelines, best practices, suggestions, tips, and industry-specific standards such as the Motor Industry Software Reliability Association (MISRA) Guidelines for the use of the C language in critical systems [4]. However, none of these sources provide a prescriptive set of secure coding standards that can be uniformly applied in the development of a software system. This conclusion is reinforced by the Secure Software Assurance Common Body of Knowledge [5], published by the DHS, which laments the *lack of public standards as such for secure programming*.

The secure coding practices proposed by CERT are based on standard language specifications as defined by official standards organizations (such as ISO/IEC) or by *de facto* standard language specifications. CERT is not an internationally recognized standards body, but it is working with organizations such as ISO/IEC to advance the state of the practice in secure coding. The ISO/IEC JTC1/SC22 WG14 international standardization working group for the programming language C, for example, has offered to provide direction in the development of the C language secure coding practices and to review and comment on drafts of the informal CERT standard.

The goal of the CERT work is to encourage organizations to develop their own coding standards to be applied on all of their projects. The organizational coding standard would codify a set of rules that are necessary (but not sufficient) to ensure the security of software systems developed in the respective programming languages [6].

A secure coding standard consists of *rules* and *recommendations*. Coding practices are defined to be rules when all of the following conditions are met:

1. Violation of the coding practice will result in a security flaw that may result in an exploitable vulnerability.
2. An enumerable set of exceptional conditions (or no such conditions) in which violating the coding practice is necessary to ensure the correct behavior for the program. One example of a rule with an exception condition is to *ensure that integer operations do not result in an overflow*. While overflow generally indicates an error condition, if the code was designed assuming module behavior then it is necessary to provide an exception for overflows resulting from this behavior.
3. Conformance to the coding practice can be verified.

"The goal of the CERT work is to encourage organizations to develop their own coding standards to be applied on all of their projects. The organizational coding standard would ... ensure the security of software systems developed in the respective programming languages."

Rules must be followed to claim compliance with a standard unless an exceptional condition exists. If an exceptional condition is claimed, the exception must correspond to a predefined exceptional condition and the application of this exception must be documented in the source code.

Recommendations are guidelines or suggestions. Coding practices are defined to be recommendations when all of the following conditions are met:

1. Application of the coding practice is likely to improve system security.
2. One or more of the requirements necessary for a coding practice to be considered a rule cannot be met.

Relationships Between Efforts

CERT representatives participating in the ISO/IEC working group on guidance for avoiding vulnerabilities through language use are coordinating their efforts. While the ISO/IEC group is working on providing language-independent guidance, the CERT effort is working on developing and consolidating the language-specific guidance that provides the foundations for the ambitious goals of OWGV.

CERT's efforts in identifying and documenting secure coding practices for C and C++ will contribute to the standardization of these practices and advance the goals of the OWGV, while the OWGV effort provides a framework for CERT language-specific efforts.

Summary

Efforts have now begun to codify secure coding practices both at the international and organizational levels. The success of the secure coding standards depends on the active involvement of members of the software development communities. To become involved in the OWGV group, visit <www.aitcnet.org/isai/> or contact the convener. To contribute to the CERT secure coding standards, go to <www.securecoding.cert.org> and review or comment on the existing content or submit new ideas for secure coding practices.◆

Acknowledgements

Thanks to Hal Burch for his information on format string vulnerabilities in languages other than C and C++ and to John Benito for his assistance in developing this article. Thanks also to reviewers Pamela Curtis, Chad Dougherty, and Fred Long.

References

1. Seacord, Robert C. Secure Coding in C and C++. Boston, MA: Addison-Wesley, 2005 <www.cert.org/books/secure-coding>.
2. ISO/IEC JTC1 SC22 WG14. Information Technology – Programming Languages, Their Environments and System Software Interfaces – Extensions to the C Library, – Part I: Bounds- Checking Interfaces. ISO/IEC TR 24731. Geneva, Switzerland: ISO, 2006 <www.open-std.orgjtc1/sc22/wg14/www/docs/n1146.pdf>.
3. ISO/IEC JTC 1/SC 22. New Work Item Proposal for Guidance to Avoiding Vulnerabilities in Programming Languages Through Language Selection and Use. 2005 <www.open-std.org/jtc1/sc22/open/n3913.htm>.
4. MIRA Limited. MISRA C: 2004 Guidelines for the Use of the C Lan-

guage in Critical Systems. Warwickshire, UK: MIRA Limited, 2004.
5. Redwine, Jr. Samuel T., Ed. Software Assurance: A Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software. Draft version 1.05. Aug. 2006.
6. Seacord, Robert C. "Secure Coding Standards." Static Analysis Summit. NIST Special Publication 500-262. Gaithersburg, MD: NIST, 2006. 14-16 <http://samate.nist.gov/docs/NIST_Special_Publication_500-262.pdf>.

Note

1. The nomenclature for international standards groups represents a hierarchical organization. The international standards committee for information technology is a JTC of two international standards-makers, the ISO and the IEC, and is therefore called ISO/IEC JTC 1. It subdivides its work among numbered subcommittees: SC 7 deals with software and systems engineering, SC 22 with programming languages, and SC 27 with computing security. SCs subdivide their work among numbered or lettered WG and OWGs.

About the Authors

James W. Moore is a 35-year veteran of software engineering in IBM and, now, the MITRE Corporation. He is an executive editor of the IEEE Computer Society's *Guide to the Software Engineering Body of Knowledge* and a member of the editorial board of the *Encyclopedia of Software Engineering*. He participates in international standardization related to software and systems engineering as well as programming languages. Moore is a fellow of the IEEE and a charter member of the IEEE Computer Society's Golden Core.

MITRE Corporation
7515 Colshire DR
H505
McLean, VA 22102-7508
Phone: (703) 983-7396
Fax: (703) 983-1279
E-mail: moorej@mitre.org

Robert C. Seacord is a senior vulnerability analyst at the CERT/CC at the SEI. He is the author of *Secure Coding in C and C++* and co-author of *Building Systems from Commercial Components* and *Modernizing Legacy Systems*, as well as more than 50 papers on software security, component-based software engineering, Web-based system design, legacy-system modernization, component repositories and search engines, and user interface design and development. Seacord also has worked at the X Consortium where he developed and maintained code for the Common Desktop Environment and the X Window System.

SEI
Pittsburgh, PA 15213
Phone: (412) 268-7608
Fax: (412) 268-6989
E-mail: rcs@cert.org

WEB SITES

BuildSecurityIn

https://buildsecurityin.us-cert.gov/daisy/bsi/home.html

As part of the Software Assurance program, Build Security In (BSI) is a project of the Strategic Initiatives Branch of the National Cyber Security Division (NCSD) of the Department of Homeland Security. The Software Engineering Institute (SEI) was engaged by the NCSD to provide support in the Process and Technology focus areas of this initiative. The SEI team and other contributors develop and collect software assurance and software security information that helps software developers, architects, and security practitioners to create secure systems. BSI content is based on the principle that software security is fundamentally a software engineering problem and must be addressed in a systematic way throughout the software development life cycle. BSI contains and links to a broad range of best practices, tools, guidelines, rules, principles, and other knowledge that can be used to build security into software in every phase of its development.

National Institute of Standards and Technology: Computer Security Resource Center

www.csrc.nist.gov

The Computer Security Division – (893) is one of eight divisions within National Institute of Standards and Technology's (NIST) Information Technology Laboratory. The mission of NIST's Computer Security Division is to improve information systems security by: raising awareness of information technology (IT) risks, vulnerabilities and protection requirements, particularly for new and emerging technologies; researching, studying, and advising agencies of IT vulnerabilities and devising techniques for the cost-effective security and privacy of sensitive federal systems; developing standards, metrics, tests and validation programs; and developing guidance to increase secure IT planning, implementation, management and operation.

Computer Emergency Readiness Team Coordination Center

www.cert.org

The Computer Emergency Readiness Team Coordination Center is a center of Internet security expertise, located at the Software Engineering Institute (a federally funded research and development center operated by Carnegie Mellon University). The team studies Internet security vulnerabilities, researches long-term changes in networked systems, and develops information and training to help you improve security.

Committee on National Security Systems

www.cnss.gov

Under Executive Order 13231 of October 16, 2001, Critical Infrastructure Protection in the Information Age, the President designated the National Security Telecommunications and Information Systems Security Committee as the Committee on National Security Systems (CNSS). The CNSS provides a forum for the discussion of policy issues, sets national policy, and promulgates direction, operational procedures, and guidance for the security of national security systems.

How a Variety of Information Assurance Methods Delivers Software Security in the United Kingdom

Kevin Sloan and Mike Ormerod
Echelon Consulting Ltd.[1]

Many in the software engineering world are already aware of the Common Criteria (CC) and the important role it plays providing confidence in the reliability of Information Technology (IT) product security claims through an independent evaluation and certification process. In the United Kingdom (UK), the CC philosophy has been adapted into a variety of pragmatic assurance techniques not just for single vendor solutions but for complete systems, sub-systems, network technologies, security architectures, and even managed service provision. How the general framework of CC can be adapted to provide assurance of security functions in software should be of interest to any person designing an assurance process for software security.

CC for IT security evaluation [1] and its forerunners have been successfully providing independent assurance of IT products since the 1980s. It has evolved as a framework for undertaking the formal independent evaluation of IT security products at either government or commercial evaluation facilities. The outcome of this process will be certification of an IT security product at a given Evaluation Assurance Level (EAL) as detailed in Table 1. The higher the EAL the more rigorous are the development and evaluation processes and the greater the confidence that the product can be trusted to perform the security functions the vendor claims, once successfully evaluated and certified.

By virtue of both the CC Recognition Agreement (CCRA) and the issue of CC as an International Standards Organization (ISO) standard (ISO 15408), CC certificates are recognized in many countries across the world.

A measure of global CC activity is demonstrated by the number of CC certificates issued. A quick check to date at <www.commoncriteriaportal.org> [2] reveals the certified product counts by category given in Table 2. *Protection Profiles* in this table refers to certificates issued to an abstract class of security functionality rather than an actual *Product Certificate*. Products are often themselves certified against a particular *Protection Profile*. One example of a class of security functionality would be role-based access control.

However, can CC be used as a framework for genesis of further assurance methods? From experiences in the UK, we believe this is the case; this is the main thesis of this article.

This article does not provide further detail on CC. For those unfamiliar with it and interested in further understanding, the full CC framework is published online at [2].

Why Is Independent Assurance So Important?

The need for IT product certification initially rose with a need for increased confidence and standardization in security products and operating systems deployed in the military domains. Safety and security requirements dictated that it was neither prudent to rely on manufacturer claims nor to allow a series of different proprietary approaches to develop functions that did not match the needs of the end user.

The natural response to this is to establish a framework of independent, functional specification and product verification. In this way those security functions can be validated as they are produced against a clear set of standardized requirements. Evaluators will verify and validate the design documents of the vendors, repeat tests using a defined sampling strategy (to ensure they produce the same results), attempt to expose flaws or vulnerabilities, and ensure the production process follows the prescribed practices and disciplines. This greatly increases the confidence in the finished product.

What Are Its Limitations?

The immediate fact to note from the

Table 1: *CC Evaluation Assurance Levels*

EAL 0	Inadequate assurance
EAL 1	Functionally tested
EAL 2	Structurally tested
EAL 3	Methodically tested and checked
EAL 4	Methodically designed, tested, and reviewed
EAL 5	Semi-formally designed and tested
EAL 6	Semi-formally verified, designed, and tested
EAL 7	Formally verified, designed, and tested

Table 2: *CC Certificates by Product Category*

Product Category	Product Certificates	Protection Profiles
Access control technology	15	1
Boundary protection technology	65	8
Data protection systems	23	0
Database systems	20	5
Digital signature technology	20	3
(Misuse) Detection technology	7	4
Integrated circuit and smartcard technology	136	22
(Cryptographic) Key management systems	18	1
Networking technology	37	5
Operating systems	49	5
Others	10	25
Total	400	79

count up of certified products given earlier (Table 2) is that most product categories are the direct components of security architectures (e.g. boundary protection), relatively few are generic IT products (e.g. operating systems and databases). This already signifies CC as a component and not a solution-focused method; a certified product is no use in isolation. Also, neither does CC meet the need for end to end assurance of network architectures and the overall assurance of business applications. Additional methods are needed to cover assurance of these solutions.

Another statistic to note is how the largest category by far (integrated circuits and smartcards, 34 percent of all certifications) are the most self-contained. These are ideal for testing from the comfort of a laboratory environment and are well defined for the rigor of CC. How do you assure a deployed solution comprising many products and which is infinitely more complex?

The main limitations of the classical CC approach can be summarized as the following:

- Practicality of evaluation reduces with complexity.
- Time, cost, and probability of an unsuccessful outcome increases with complexity and level of assurance required.
- It is impossible to define complex systems to the depth necessary to subject them to high assurance levels.
- It is difficult to apply to a collection of commercial off-the-shelf products due to the complexity introduced by combined permutation of configurations and competing proprietary interests (version 3.1 of CC now adds the concept of composition assurance profile to enable this form of evaluation).
- It is a product-focused credential; it cannot assure particular installations of that product outside of the laboratory environment or assure an overall service delivery that includes the product.
- The specialist nature of the products means it often is difficult for vendors to recover the cost of evaluation through volume; evaluated versions can have a high-cost premium.
- Government sectors know how to apply assured products by virtue of their security policies, but industry has no benchmark to apply them. In this environment, it is hard to build a business case for them.
- Complex evaluations take a long time; often, the certified version may be a product generation behind the latest version on the shelf once it becomes certified.
- Evaluated configurations are often limited in scope, not matching real-world implementations.
- Evaluation is not perfect; there will still be security patches.

Alternative Approaches

What other approaches are there to assure the security of delivered solutions? There are alternatives for providing confidence in software production processes, such as the Capability Maturity Model Integration (CMMI®) [4]. There are also holistic control-based approaches to information security that can be applied at the organization and business process level including ISO27001/ISO17799 [5,6] and Control Objectives for IT (COBIT) [7].

At the other extreme, there are both tools and experts available to provide vulnerability assessments and penetration tests of deployed solutions. The *tried and trusted* method is also still abundant in industry in which products are only incorporated after extensive proving within a customer environment.

Some product groups also have specialist watchdogs that provide constantly updated information on product quality. For instance, Virus Bulletin [8] provides a constantly updated profile of the performance of all anti-virus products on the market. ICSA labs [9] provides a subscription-based, broad ranging commercial product security list that was tested, and using their own proprietary method, West Coast Labs have established a proprietary independent product certification scheme (Check-Mark) [10].

However, none of the alternatives offers the depth and specificity of the CC process. The obvious approach is to customize the CC to the alternative needs to derive a range of information assurance (IA) techniques that are tailored to fit the specific assurance needs of each circumstance. CC especially lends itself to the production of derivative methods due to its modular nature and structured philosophy; it can be used as a toolbox to provide derivative assurance methods.

What's Happened in the UK

This is exactly what has happened in the UK: The Communications-Electronics Security Group (CESG) is the UK's national technical security authority. It has led initiatives to derive a selection of IA methodologies driven by customer demand (inevitably clients from the UK government sector). It is the main driving force behind all UK government technical security initiatives.

The remainder of this article is dedicated to this range of methods beyond the conventional CC evaluation process. These include the following:

- System level evaluations (SYS) [11].
- Fast-track assessments (FTA) [12].
- IT security health checks (CHECK) [13].
- CESG Assisted Products Scheme (CAPS) [14].
- Central Sponsor for Information Assurance (CSIA) Claims Tested Mark (CCTM) [15].

The SYS, FTA, and CHECK methods can augment laboratory based verification with assessment at the point of installation for a particular project. Also, the UK now has the CCTM scheme to test the manufacturer security claims of shrink wrapped products and services, using a single-pass, low-cost method; this proves suitable for assessing products and services which have so far eluded formal evaluation due to their polymorphic nature (including anti-virus software and content filters). It also moves away from open ended and unaffordable iterative approaches that risk never having a successful outcome to fixed assessment projects with known milestones, costs, and end products.

The Key Players

Apart from CESG, referred to above, the other UK organizations that play a key part in delivering IA requirements to the UK Government sector include the following:

- **Cabinet Office.** The Cabinet Office is the source of most political initiatives that drive UK public sector interest in information systems and IT. For instance, the UK has a requirement to provide all government services to the citizen online by the end of 2007. The Cabinet Office also owns UK government security policy.
- **CSIA.** CSIA is part of the Cabinet Office and takes a pan-civil government responsibility for accreditation of public sector information systems and IT. This remit extends from all central government departments through to local authorities and government agencies.
- **National Infrastructure Security Coordination Center (NISCC).** NISCC performs in the UK as part of the functions of homeland security. NISCC is part of the wider security services for ensuring the security and

® Capability Maturity Model and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

availability of the critical national infrastructure. This extends beyond government to privatized utilities and critical services. NISCC also promulgates technical security advice and is also the central point for Unified Incident, Response and Alerting Scheme which is the UK government's Computer Emergency Response Team (CERT).
- **Ministry of Defense (MOD).** MOD is the largest UK consumer and investor in IA services and is, therefore, the key stakeholder. Naturally, they need to apply full rigor of formal processes to the assurance of systems and have helped to shape the methods in use today.
- **Department of Trade and Industry (DTI).** DTI has a role in advising industry on information security matters, especially for government contractors. It has been a key player in the past in promotion of adoption of standards in use by government to the commercial sector and in the standardization at both the national and international levels.
- **UK Accreditation Service (UKAS).** UKAS oversees all UK government sponsored certification services in the UK including the certificate or testing laboratories.

All of the above have had important roles in definition and practice of the standards and methods discussed in this article.

SYS
SYS is a clear CC derivative developed by CESG to meet MOD requirements for assurance of whole solutions. Typical MOD IT systems incorporate multiple products from multiple vendors and may extend from a system deployed in a single data center to an enterprise solution covering many sites.

Modeled along the assurance levels, the equivalent SYS2 to SYS4 levels provide a measure of equivalence to EAL2 to EAL4. It excludes those elements of CC that are specific to individual products and vendors and focuses on system level and end to end testing. It was formalized as a UK IT Security Evaluation Criteria (ITSEC) method in 2002.

It was the first method to reduce the iterative approach of CC and the pure *pass/fail* outcome to a more risk based approach in which the test report provides information on residual flaws and vulnerabilities; it is then for the system accreditor to accept the overall solution as is, to call for remedy in specific areas, or to build in compensating controls. This produces an approach that is more predictable in terms of project costs and timescales.

FTA
CESG, on realizing the benefits of the system-level approach and its broader applicability to all UK government projects then embarked on definition of a more generally usable *fast track* scheme that could be applied to specific products and for deployment of specific components. It was launched as a CESG IA service in 2001.

Here the emphasis is for an entirely predictable time to evaluate and for the cost to be known at the outset. Again the focus is on deployment of the product in a specific project environment, which can be a benefit over the more generalist approach of CC.

"CC especially lends itself to the production of derivative methods due to its modular nature and structured philosophy; it can be used as a toolbox to provide derivative assurance methods."

It is designed to be at lower cost and therefore more within the budgets of the smaller government departments.

The output is an assessment report rather than a certified product; it is developed with the intention that it provides assurance that the product is fit for the purpose in the sponsor's desired deployment.

CHECK
The CHECK scheme is not strictly a derivative of CC. CHECK is another service from the CESG stable that has been operating since 1998. This scheme focuses on providing UK government trained *ethical* penetration testing personnel for undertaking penetration testing and vulnerability testing on deployed UK government networks and solutions. These test personnel undergo a stringent annual assault course examination at CESG to ensure their skills remain current. Companies that maintain the test personnel are given a green light status by CESG which allows them to tender for CHECK contracts.

The CHECK service is mandated for all UK government projects that involve the connection of sensitive UK government information systems to the Internet. An important recent development of this scheme (which originally focused on network testing) is a new application testing service that focuses on assessment of web enabled multi-tier applications and the vulnerabilities that can be exposed purely by the interaction of multiple software packages, how they are configured, and the presence of underlying bespoke application code.

CAPS
The CAPS scheme is the UK equivalent of the US Federal Information Processing Standard (FIPS)-140 encryption product evaluation standard. CAPS and FIPS-140 are formal means of evaluating cryptographic hardware and software products in a similar way to which CC evaluates IT products.

Cryptography is typically excluded from CC evaluations due to national sensitivities and import/export laws: cryptography is judged as having a *dual use* (military) application and therefore useful to rogue administrations.

UK government departments are required to protect UK classified information with CAPS approved products.

UK vendors of encryption technology enter into a partnership with CESG to engineer UK specific algorithms into their products. Once approved, these products become part of the UK catalogue of approved encryption products.

CSIA CCTM
CCTM is the latest scheme targeted with the widest possible applicability. This has been championed by the CSIA, with technical assistance of CESG. It is an entry-level assurance method; we have heard it described as *what EAL1 should have been.*

It is intended to be an affordable credential for all forms of security products and services. It is a single pass method that concentrates on validating vendor product functional security claims. The scheme was launched with pilot assessment in 2005 and is now fully in effect. Twelve security products and one service have so far been assessed under the scheme and awarded the mark. The single service assessed so far is the Messagelabs e-mail scanning service which exists between the UK government secure intranet and the Internet.

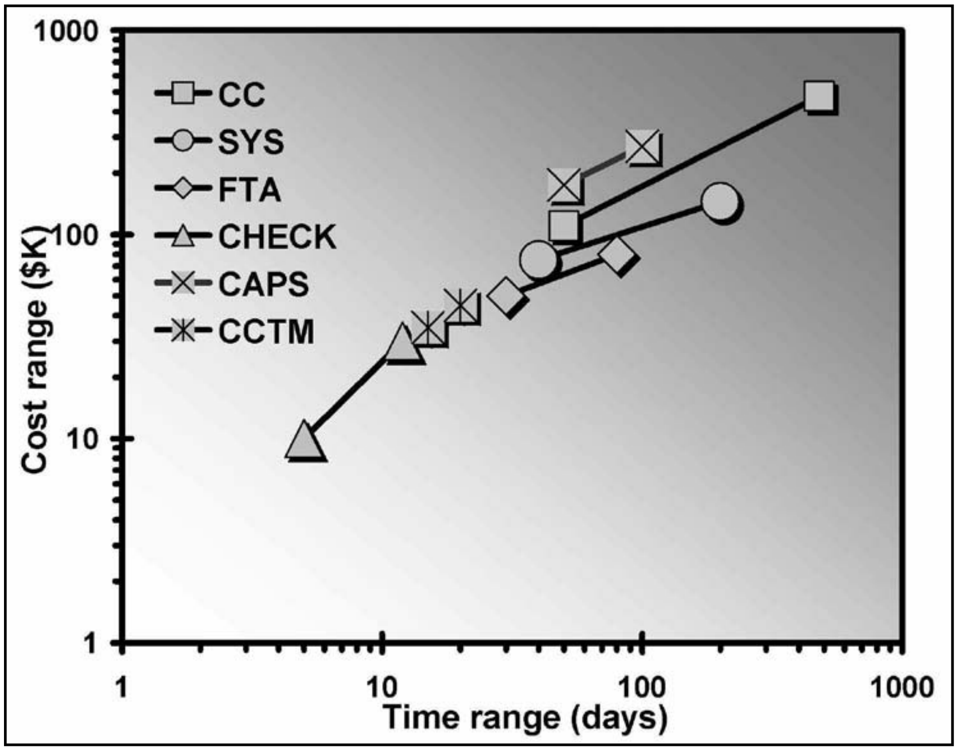


Figure 1: *Comparison of Assurance Method Ranges of Cost and Time*

The process is a simple single-pass process with limited functional testing that starts with a vendor provided IA Claims Document that specifies the security functionality of the product or service to be assessed.

Designed to be fast and at very low cost, it provides a baseline UK government approved badge. All new security products and services will need this as the minimum credential for inclusion in UK government procurement catalogues.

Table 3: *Information Assurance Method Requirements Fit*

Assurance Requirement	CC	SYS	FTA	CHECK	CAPS	CCTM
Certified product mark	✓				✓	✓
Mutual recognition	To EAL 4					
High assurance	✓				✓	
Medium assurance	✓	✓	✓		✓	
Low assurance				✓		✓
Pass/fail outcome	✓			✓	✓	✓
Risk enumeration		✓	✓	✓		
Flexible re-use	✓					✓
Project specific		✓	✓	✓		
Deployment specific		✓	✓	✓		
End to end		✓		✓		
Bespoke specific		✓	✓	✓		
Multi-product/vendor	In V 3.1	✓		✓		
Polymorphic products				✓		✓
Architecture/system		✓		✓		
Service delivery						✓
UK national requirements		✓	✓	✓	✓	
Deployment testing		✓		✓		
Open ended/iterative	✓				✓	
Time bounded		✓	✓	✓		✓
Low cost			✓			
Very low cost				✓		✓

Applicability of the Approaches

The coverage of the UK methodologies can be best illustrated by a comparison of the typical indicative minimum to maximum cost and time ranges for each of the derivative methods (SYS, FTA, CHECK, CAPS, and CCTM) displayed against the classical CC ranges. These are shown in Figure 1. Note that this diagram has logarithmic scales.

The cost ranges only relate to the evaluation and assurance processes themselves. The more rigorous methods impose necessary additional development costs on the vendor that are not shown and that also depend on the complexity of the component or system to be evaluated or assured.

So these practical considerations derived a set of IA approaches that are tailored to the demands of the UK consumers. This now provides a full spectrum of methods that can be applied to meet real-world assurance activities arising from information age projects in the UK.

Table 3 provides a summary of the characteristics and fit of these different methods according to the different project needs. This clearly demonstrates the need for a selection of methods to cover the full range of assurance requirements that are needed to address an increasingly diverse UK market.

How Can This Be Exploited Outside of the UK?

Alas, the methods outlined in this article are UK specific and are outside of the scope of CCRA. Plus, with the exception of CCTM, they are not focused on re-usable product certification but on assurance and accreditation requirements for specific UK projects (with there being some scope within the UK for re-use of accredited system designs and architectures). However, the methods of the CC derivatives are published in the public domain.

It is, of course, useful knowledge to international companies that wish to sell products and solutions to the UK government market.

However, the principal shown is that it is possible to derive efficient and successful assurance processes based on the CC model. CC can be used as a resource to build methodologies for enhancing the security in software engineering even if a product certificate is not the ultimate requirement. This can have universal application and can reap the same benefits experienced in the UK of timely deliveries, cost effective assurance regimes, and increased confidence in the security of the end product.

Looking Forward

It is possible to envisage further assurance methods that will be required, in the UK or elsewhere. The CHECK scheme extension into the testing of deployed applications demonstrates that there is a need to encompass all software in the overall assurance framework. Confidentiality, integrity and availability are increasingly protected not by isolated security products but by different aspects of the whole solution.

One emerging development is that CESG has commenced a project to rationalize the SYS and FTA methods into a *tailored assurance* framework that will allow specific programs of work to use CC derivative methods fitting to their exact need. This is a natural progression of the philosophy outlined in this article. Perhaps this is something that will be taken forward to future generations of CC and ISO 15408.

Conclusions

Is it right to compromise on the original design of CC? This article has demonstrated several derivative methods that reduce its complexity and that focus on risk management and acceleration of the development process. It is important to realize that CC is already a risk management method; evaluators do not repeat every vendor test, nor do they inspect every line of code. They adopt sampling approaches to assure the quality of the vendor product, just as in any risk based audit. It is also a benefit to be able to expand the use of elements the CC methodology from the limited boundaries of individual software, hardware, and firmware products to more open ended solutions and entire architectures.

We argue that the derivatives only extend the risk management concept to allow application of its principals in wider circumstances and to meet real-world business needs. We strongly believe in the founding concepts of CC and the importance of independent assurance of secure solutions. By recognizing the diversity of requirements and the flexibility of the framework to serve them, it will allow the CC method to meet its vision of universal recognition of this value in the years to come.◆

References

1. ISO/IEC 15408. "Evaluation Criteria for IT Security." ISO 2005 <www.iso.org>
2. Common Criteria Portal <www.commoncriteriaportal.org>.
3. UK ITSEC. CESG/DTI 1990 <www.cesg.gov.uk>.
4. Carnegie Mellon University. Capability Maturity Model® Integration (CMMI®). CMU Software Engineering Institute <www.sei.cmu.edu/cmmi>.
5. ISO/IEC 27001:2005. Information Security Management Systems – Requirements <www. iso.org>.
6. ISO/IEC 17799:2005. Code of Practice for Information Security Management. 2005 <www.iso.org>.
7. IT Governance Institute/Information Systems Audit and Control Association. "The Control Objectives for Information and Related Technology." (COBIT 4.0) 2006 <www. isaca.org>.
8. "Independent Malware Advice." Virus Bulletin Ltd. 2006 <www.virusbtn.com>.
9. ICSA labs. Online portal <www.icsa.net>.
10. "Checkmark Certification and Product Testing." West Coast Labs <www.westcoastlabs.org/checkmarkcertification.asp>.
11. UK ITSEC Scheme Certification Body. UK IT Security Evaluation and Certification Scheme – SYSn Assurance Packages Framework (Issue 1.0). 2002 <www.cesg.gov.uk>.
12. CESG. "Fast Track Assessment Service: Overview." CESG. 2001 <www.cesg. gov.uk>.
13. CESG. "CHECK Service Provision Guidelines (Vers. 7.0)." 2002 <www.cesg.gov.uk/site/check/index.cfm>.
14. CESG. "Assisted Products Scheme." <www.cesg.gov.uk>.
15. CSIA. "CSIA Claims Tested Mark Scheme - Description of the Scheme (Vers. 2.2.0)." CSIA <www.cabinetoffice.gov.uk/csia/documents/pdf/cctm/scheme_description.pdf>.

Note

1. Echelon Consulting Ltd. (UK) is registered as a legal business entity in the UK and has no business connection with 2004-2006 Echelon Corporation of San Jose, CA.

About the Authors

Mike Ormerod is a qualified ISO15408 lead evaluator and consultant with 17 years evaluation experience and has been in the IT industry for 22 years. He has in depth knowledge in security evaluation and CCTM assessments. Ormerod has had a variety of customers in the UK government and private sectors, as well as overseas customers expanding to the far east. He is currently busy establishing the Echelon Test Facility, based in London, which is currently undergoing the UKAS certification process for Test Laboratory status.

Echelon Consulting Ltd.
Echelon House
93 Fleet RD
Fleet
Hampshire
GU51 3PJ
United Kingdom
Phone: +44(0)1252 627799
Fax: +44(0)1252 626509
E-mail: mike.ormerod@echelonltd.com

Kevin Sloan is a principal security consultant and the Echelon Consulting Ltd. (UK) Technical services manager. He has a background in electronics, microprocessor applications, firmware, software, and data communications, and specializes in information security. He has more than 26 years experience in the IT industry with information and communications security experience spanning over the last 18 years. Sloan is qualified by CESG to undertake IA activities for many UK government customers and has been a key player on several significant UK eGovernment projects. He has broad experience with commercial clients in the UK and overseas.

Echelon Consulting Ltd.
Echelon House
93 Fleet RD
Fleet
Hampshire
GU51 3PJ
United Kingdom
Phone: +44(0)1252 627799
Fax: +44(0)1252 626509
E-mail: kevin.sloan@echelonltd.com

High-Leverage Techniques for Software Security

Idongesit Mkpong-Ruffin and Dr. David Umphress
Auburn University

Software security addresses the degree to which software can be exploited or misused. Software development approaches tend to polarize security efforts as being reactive or proactive; a blend of both approaches is needed in practice. Three categories of tools provide such a blend: threat modeling, risk analysis, and security assessment and testing. These tools provide leverage as they are currently in use as quality assurance methods and can be modified with relatively little effort to address security.

Software security deals with the ability to protect software and its underlying systems from being exploited by unauthorized users and from mishaps from authorized users. This includes safeguarding against direct attacks [1], detecting and preventing indirect attacks that take advantage of software defects, removing defects, and preserving the intrinsic value (such as the inherent intellectual property) of the system.

Approaches to ensure that software systems are secure fall along a continuum ranging from reactive to proactive techniques. Reactive proponents tend to favor a post-facto approach – applying security measures to software after it has been developed. This is based on the opinion that it is not possible to protect against every single conceivable threat. This philosophy is articulated as *if security is the absence of risk, then we will never get a system that is both secure and useful. There is a need to balance risk and control* [2]. Reactive approaches hinge on three assumptions. First, systems that are exhaustively assessed for security become hard to use and the cost in time and effort to produce and use such systems defies the economics of software development. Second, risk handling should be deferred until a problem actually occurs. Third, tools to handle problems need to be in place to handle issues when they arise [2].

The proactive approach, on the other hand, requires that security measures be implemented during the software life cycle as part of the development process, so that fielded systems require little or no patching. This is based on the premise that if software is built with security in mind then vulnerabilities will be addressed early in the development cycle. To do so, proponents of proactive security measures require that architectures and designs that actively promote security be created and that risk management be applied throughout the development process. Put bluntly, this is the philosophy of *building things right, designing for security, analyzing security over the whole life cycle, and coding securely* [3].

In practice, both approaches are necessary to create secure software. Since systems are often a composite of unreliable parts, putting those parts together to ensure security is an engineering problem, and as such, software development should be approached with the attitude that secure and effective solutions can be created even when some of the materials used are flawed [2].

"The purpose of threat modeling is to examine the security of a software component from the perspective of what types of attack are likely to take place on it."

Recent media attention on the frequency of security patches that are required by popular software packages may lead developers to think that software security is an overwhelmingly complex issue. While security is difficult, it is not impossible. There are three specific high-leverage techniques that can be used, such as threat modeling, risk analysis, and software assessment and security testing. Each of these is well within the reach of the skills of most developers.

Threat Modeling

The purpose of threat modeling is to examine the security of a software component from the perspective of what types of attack are likely to take place on it. Traditionally, the practice of using graphical software representations, such as Unified Modeling Language (known as UML), focuses on describing what is expected of a system, not what is unexpected. Threat modeling augments this approach by requiring developers to consider ways a component might be misused based on past history. It helps developers think beyond standard features and consider negative or unexpected events. Arising from threat modeling are misuse and abuse cases which address abnormal behavior. Extending use-cases to include *misuse*-cases that depict side-by-side what behavior should be supported and what should be prevented has been proposed [4]. Threat modeling helps developers view the software component from the perspective of an attacker, thus bringing security to the forefront during all phases of development.

Threat modeling begins with a catalog of agents and attacks that have been carried out on other systems. There are vulnerability databases such as the National Vulnerability Database (NVD) [5] and Open Source Vulnerability Database (OSVDB) [6] that incorporate publicly available vulnerability resources and would make a good starting point in documenting these agents and attacks. At its most fundamental, the catalog consists of well-known vulnerabilities, such as problems arising from boundary conditions, intersystem communication, system assumptions, etc. Other vulnerabilities are added as they are observed.

The threat model specific to a software component – from which use and misuse cases are constructed – is constructed in the following three steps [7]:

1. A behavior model of intended functions is defined based on the functional requirements.
2. Based on the behavior model, decisions on potential misuse/abuse or anomalies of the intended functions that would violate any of the security goals are made.
3. Mitigations for misuse or anomalies in the threat model are specified.

The steps are frequently carried out iteratively, with the derived security models built with different levels of detail.

Tools and methodologies that are helpful in threat modeling are attack trees [8], attack nets [9], and attack pattern matching [10]. These methodologies approach threat identification at different levels of abstraction. Attack trees diagram the steps an attacker would take in completing his/her objective [8]. Attack trees tend to be most effective in smaller applications. An attack net, an extension of an attack tree, is a Petri net used to represent complex security threats with variation within attack patterns [9]. Attack nets are not meant to model the actual behavior of a system/component when an attack happens but are used to organize the development of probable attack scenarios. Concurrency and attack progress are modeled as tokens, and intermediate and final objectives are modeled as places, while commands or inputs are modeled as transitions [9]. Attack pattern matching processes such as Security Analysis for Existing Threats [10] and Microsoft's Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege (STRIDE) [11] approach, are designed to match attack patterns to system designs. These processes are used at the design level to provide a higher level of abstraction so that the evolving system design can incorporate security measures to the process.

Risk Analysis

Risk is the probability that an event with a negative impact will occur. It is determined by factors such as the ease of executing an attack, the attacker's motivation and resources, a system's existing vulnerabilities, and the cost or impact in a particular business context [1]. Risk analysis deals with the way threats are described. It takes into account the impacts, the probable consequences, and the probability and frequency of occurrence of each threat [1, 12]. It helps in determining ways to alleviate identified risks by providing selection criteria for safeguards and other means of preventing or lessening identified risks to a level that is considered acceptable.

The activities that are done during risk analysis are determined by the security requirements received during requirements and design phase analysis. Common risk analysis activities include risk identification, assessment, characterization, communication, mitigation, and risk-specific policy definition [12]. Techniques such as asset valuation, quantitative risk analysis, and qualitative risk analysis are used in risk analysis to gather required information so that security-relevant design specifications can be created [13]. The information gained from risk analysis is used while choosing tools and mechanisms for the security design process. The value of assets and the cost of attacks are compared with the costs of tools and mechanisms in order to ensure the chosen tools and mechanisms are appropriate and proportionate to the risk to which the application or system is exposed.

"Quantitative risk analysis is used to identify the key risk elements and the value associated with identified risk."

Asset valuation is the process used to determine the worth of an asset. It examines information not only of the hardware and software pertinent to the system but also personnel and other physical assets. The value of the asset is made up of its inherent value and the short and long term impacts and consequences of its compromise [14]. This aids in justifying proposed mitigation to stakeholder, legal and other regulatory requirements [1, 12].

Quantitative risk analysis is used to identify the key risk elements and the value associated with identified risk. This information allows for the estimation of potential loss and the ability to analyze potential threats. Quantitative risk analysis is used to compute what is known as the Annual Loss Expectancy (ALE):

ALE = Single Loss Expectancy (SLE) x Annualized Rate of Occurrence (ARO)

where,

ARO is the frequency of threat per year, SLE is the asset value x the exposure factor (EF), and EF is the percentage of asset loss caused by the potential threat.

The values garnered from the quantitative analysis can then be ranked and decisions can then be made based upon the information.

Lastly, qualitative risk analysis is used to discover the threats and vulnerabilities that apply to different identified scenarios. Safeguards and countermeasures that reduce or prevent the probability and effect of occurrence are identified, based on the threats and vulnerabilities found [12]. Most qualitative risk analysis methodologies assign weight and values to categories such as probability of occurrence, impact, exposure, and cost. Higher values are assigned to categories that are assumed to be of greater importance, thereby reflecting their impact in the overall priority score.

The information gathered during risk analysis is then used in the rest of the development cycle. Fault-injection methods and security tests are driven by the vulnerabilities and risks discovered and annotated during risk analysis.

Software Assessment and Security Testing

Testing traditionally involves exercising an application to see if it works as it should. In contrast, security testing entails identifying and removing vulnerabilities that could result in security violations. It also validates the effectiveness of security measures that are in place [15].

Most of the testing methodologies used fall into one of two categories: black-box or white-box testing. Black-box tests are those whose data are derived from the specified functional requirements in which attention is not given to the final program structure [16, 17]. Commonly used black-box testing approaches for software security are penetration, functional, risk-based, and unit testing.

White-box tests are those tests and assessment activities where the structure and flow of the software under review are visible to the tester. Testing plans are made based on the details of the software implementation and test cases are based on the program structure [15, 16, 17]. Commonly used white-box assessment approaches that can assess security are source code analysis and profiling.

The method by which security assessment and testing is carried out depends on the perspective of the tester relative to the software component. Test cases that are constructed based on functional requirements without regard to specific knowledge about software internals are known as black-box tests; test cases that take advantage of internal structure are known as white-box tests. Often, the information gathered during risk analysis is used to develop white-box and black-box test cases. In particular, flaws identified during risk analysis can be purposely added to a software component to forcibly change the program state and demonstrate the effects of a successfully exploited vulnerability. This approach, known as fault injection, allows for absolute worst-case prediction [18]. It gives an insight into predictive measures such as mean-time-to-hazard, minimum-time-to-hazard, and mean-time-to-failure; all of which quantify risk.

Three approaches are commonly taken to test the security of a component in a black-box fashion. Risk-based testing demonstrates that security functionalities work as intended [19]. Penetration testing examines the ease with which a component can be infiltrated. Unit security testing assumes that adversaries will take a two-stage approach to attack: First, they get access to the software, then second, control the software after access. As such, the assumptions that developers make about the environment and incorporate into the components should be checked at the unit testing level. Attack trees have been used by many as a method for identifying and modeling security threats, especially those that involve many stages for implementation [20].

Two high-leverage white-box techniques for assessing and validating security are source code analysis and profiling. Static analysis tools are used to look at the text of a program while it is not executing so that it can discover vulnerabilities within the program. A fixed set of patterns or rules are used as basis for scanning the source code. For example, many vulnerabilities are known to come from reusable library functions such as stropy() and stat (); so, a static analyzer could scan the programs to see if they contain any calls to those functions. The result of the source code analysis aids in the development of test cases and gives a good perspective of the security posture of the application. White-box testing should be used to verify that the potential vulnerabilities uncovered by the static analysis tool will not lead to security violations [21].

Profiling tools enable the tester to observe the performance of an application while it is running. This provides insight into where performance bottlenecks may be occurring. It also enables the tester to see and understand the sequence of function calls and the time spent in different areas of the software, and thereby brings it to the open areas of vulnerability that are not apparent when using static code analyzers [12].

"Software security demands a balance of reactive and proactive measures, and it requires that more time be spent in determining the risks that can or will affect the system."

Although security aspects of software should be tested, it is also important to understand that security is not just a function that can be checked off but is an emergent property of the application. In other words, this would be analogous to saying that *being dry* is an emergent property of being inside a tent during a rainstorm. The tent only keeps a person dry if the poles are made stable, vertical, and able to support the weight of the wet fabric; the tent also must have waterproof fabric (without any holes) and be large enough to protect all those who want to remain dry. Lastly, everyone must stay under the tent the whole time it is raining. So, although having poles and fabric is important, it would not be enough to say *the tent has poles and fabric, thus it keeps one dry!* [22].

Conclusion

To develop software systems with security as an emergent feature entails that the high leveraged techniques discussed be incorporated into the whole software development life cycle. Threat modeling that drives risk analysis begins with the garnering of requirements and use cases. Risks generated from the threat modeling activities act as a barometer for design, development of tests, and development of rules for software code assessment and as one of the benchmarks for testing.

Software security demands a balance of reactive and proactive measures, and it requires that more time be spent in determining the risks that can or will affect the system. Software systems have to be designed from a high enough level of abstraction with security of the system as an emergent feature of the system in question. The processes utilized to create secure systems need more refinement so that the ubiquity of software is not hampered by inherent insecurity due to poor design.◆

References

1. Verdon, Denis, and Gary McGraw. "Risk Analysis in Software Design." IEEE Security and Privacy 2.4 (2004).
2. Cheswick, B, Paul Kocher, G. McGraw, and A. Rubin. "Bacon Ice Cream: The Best Mix of Proactive and Reactive Security?" IEEE Security and Privacy 1.4 (2003).
3. McGraw, Gary. "Building Secure Software: Better Than Protecting Bad Software." IEEE Software 5.7 (2002).
4. G. Sindre, and A.L. Opdahl. Templates for Misuse Case Description. Proc. of the Seventh International Workshop on Requirements Engineering, Foundation for Software Quality (REFSQ 2001), 4-5 June 2001, Switzerland.
5. United States. Department of Homeland Security (DHS). National Vulnerability Database 7 Dec. 2006 <http://nvd.nist.gov/>.
6. OSVDB. Open Source Vulnerability Database. 8 Dec. 2006 <www.osvdb.org>.
7. Dianxiang Xu, and Kendall Nygard. "A Threat-Driven Approach to Modeling and Verifying Secure Software." Proc. of the 20th IEEE/ACM International Conference on Automated Software Engineering ASE, Nov. 2005, Long Beach, CA. New York: ACM Press, 2005.
8. Schneier, B. "Attack Trees: Modeling Security Threats." Dr. Dobb's Journal Dec. 1999.
9. McDermott, J.P. "Attack Net Penetration Testing." Proc. of the 2000 Workshop on New Security Paradigm,

Sept. 2000. Ballycotton, County Cork, Ireland. New York: ACM Press, 2000.

10. Gegick, M., and L. Williams. "Matching Attack Patterns to Security Vulnerabilities in Software-Intensive System Designs." Proc. of the 2005 Workshop on Software Engineering For Secure Systems; Building Trustworthy Applications, 15-16 May 2005, St. Louis, MS. New York: ACM Press, 2005 <http://doi.acm.org/10.1145/1083200.1083211>.
11. Hernan, Shawn, Scott Lambert, Tomasz Ostwald, and Adam Shostack. "Threat Modeling – Uncover Security Design Flaws Using The STRIDE Approach." MSDN Magazine Nov. 2006.
12. Steel, Christopher, Ramesh Nagappan, and Ray Lai. Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management. Prentice Hall, 2005.
13. McGraw, Gary. Software Security: Building Security In. Addison-Wesley Professional, 2006.
14. United States. Department of Commerce. An Introduction to Computer Security – The NIST Handbook. NIST, 1995.
15. Pan, Jiantao. "Software Testing – 18-849b Dependable Embedded Systems." Carnegie Mellon University, 1999 <www.ece.cmu.edu/~koop man/des_s99/sw_testing>.
16. Howard, Michael, and David C. LeBlanc. Writing Secure Code. 2nd ed. Redmond, WA: Microsoft Press, 2002.
17. Hetzel, William C. The Complete Guide to Software Testing. 2nd ed. Wellesley, MA: QED Information Sciences, 1988.
18. Voas, Jeffrey M., and Gary McGraw. Software Fault Injection: Inoculating Programs Against Errors. New York, NY: John Wiley & Sons, 1998.
19. Michael, C.C., and Will Radosevich. "Risk-Based and Functional Security Testing." DHS. BuildSecurityIn Portal <https://buildsecurityin.us-cert.gov/portal/article/bestpractices/security_testing/overview.xml#Risk-Based-Testing>.
20. Schneier, B. Secrets and Lies: Digital Security in a Networked World. New York: John Wiley & Sons, 2000.
21. Janardhanudu, Girish. "White Box Testing." DHS. BuildSecurityIn <https://buildsecurityin.us-cert.gov/portal/article/bestpractices/white_box_testing/overview.xml>.
22. Hope, Paco, Gary McGraw, and Annie I. Anto'n. "Misuse and Abuse Cases: Getting Past the Positive." IEEE Security and Privacy 2.3 (2003).

About the Authors

Idongesit Mkpong-Ruffin is a computer science and software engineering doctorate student at Auburn University. She has a Bachelor of Science in computer information Science from Freed-Hardeman University, a Master of Business Administration from Tennessee State University and a Master of Science in computer information science from Troy University, Montgomery campus.

Department of Computer Science and Software Engineering
Auburn University
107 Dunstan Hall
Auburn, AL 36849-5347
Phone: (334) 844-7001
E-mail: mkponio@auburn.edu

David A. Umphress, Ph.D., is an associate professor of computer science and software engineering at Auburn University. He has worked over the past 25 years in various software development capacities in both industry and academia. He is also an Air Force reservist, currently serving as a researcher for the College of Aerospace Doctrine, Research and Education, Maxwell AFB, Alabama. Umphress is an Institute of Electrical and Electronics Engineers certified software development professional.

Department of Computer Science and Software Engineering
Auburn University
107 Dunstan Hall
Auburn, AL 36849-5347
Phone: (334) 844-6335
E-mail: umphrda@eng.auburn.edu

Baking in Security During the Systems Development Life Cycle

Kwok H. Cheng
Booz Allen Hamilton

Vulnerabilities in software that are introduced by mistake or poor practices pose a serious threat. Combating threats in today's electronic environment requires a methodical approach in building security into software from the ground up, or baking in security as some of us refer to it. The absence of a planned approach opens the possibility for application flaws which adversaries could potentially exploit. Exploiting application flaws is a likely scenario, considering a Gartner report that states that 75 percent of successful attacks are targeted towards vulnerabilities at the application layer [1].

Traditional systems development lifecycles (SDLCs) used to develop information systems generally exclude security activities. Without explicitly defining security activities during systems development, security may not be completely planned for or even considered. Integrating security activities into the SDLC helps address this issue. Security activities can be included in phases such as requirements, design, build, or test to ensure that governance or industry best practices are satisfied, and that the goal of the system is achieved without conflict.

The Benefits of Integrating Security Into the SDLC

Bolting on security post-development is no longer sufficient in delivering systems on time, under budget, and with the proper level of protection in place. Addressing the issue, the Computer Emergency Readiness Team Coordination Center (CERT/CC) states the following:

> … many security incidents are the result of exploits against defects in the design or code of software. The approach most commonly employed to address such defects is to attempt to retroactively bolt on devices that make it more difficult for those defects to be exploited. This is not a solution that gets to the root cause of the problem and threat. [2]

It is estimated that bolting-on security post-development costs roughly three times more than the cost of built-in security. According to Gartner, if 50 percent of vulnerabilities were removed before production of purchased and internally developed software, enterprise configuration management (CM) costs and incident-response costs could be reduced by 75 percent each [3].

Building in security ensures proper and cost-effective protection. Assets that are identified and categorized can be more appropriately protected in terms of adequacy and cost. For example, when risk analysis is performed during the requirements phase, security risks may be identified which translate into security requirements.

Approaches for Integrating Security Into the SDLC

Regardless of the SDLC model used (e.g. waterfall, spiral, Rational Unified Process), the SDLC represents a phased approach to the development of a system. An appropriate model should facilitate the reanalysis and validation of the plans, requirements, and design at multiple points throughout its life cycle. Whether regarded as a phase or discipline, an SDLC is composed of several common groupings of activities: requirements, design and build, test and deploy, operations and maintenance, and disposal, with full lifecycle support activities such as risk management, CM, and training. Security can be integrated into these different points of the SDLC independent of the model.

Figure 1 describes each phase or discipline of an SDLC with the associated security activities.

Full Life-Cycle Support Activities

Risk Management

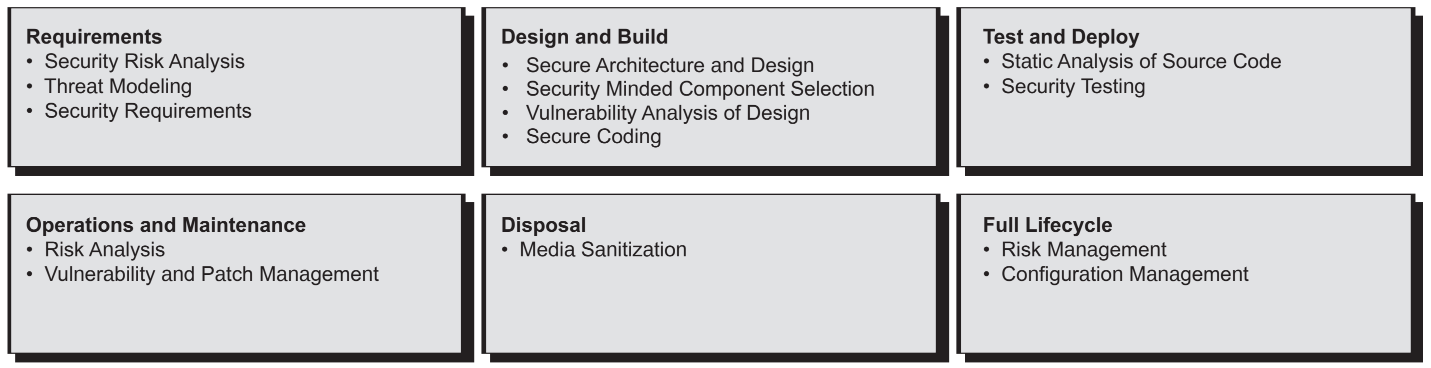
Risk management includes performing security risk analyses at different points of the life cycle. Security risk analysis serves to identify and mitigate security-related risks.

The results of the risk analysis feed into the risk management process of identifying, controlling, and eliminating or minimizing uncertain events that may affect the system. Risk analysis should be repeated iteratively throughout the system's life cycle as different activities allow opportunities to identify new or changing risks. For instance, as the project progresses forward and activities shift from requirements development to high-level system design, additional information will be uncovered about the application. This new information may reveal risks not previously identified such as use of vulnerable components or a flawed authentication model. We also know that changes to design during the build phase are almost always certain to occur. That is why it is important to also perform a risk analysis on the system after it has been built.

CM

Inaccurate or incomplete CM may enable malicious developers to exploit the shortcomings in the CM process in order to make unauthorized or undocumented changes to the software. Lack of proper software change control, for example, could allow rogue developers to insert or substitute malicious code, introduce exploitable vulnerabilities, or remove or modify security controls implemented in the software. Good CM practices also prevent the introduction of unintentional flaws into software code. For example, a developer makes a seemingly harmless modification to the application's interface before deployment and is able to bypass the CM process. This change unintentionally gives normal, restricted users elevated privileges to view information they normally would not be allowed to access. Since the CM process was bypassed, this change was not analyzed or tested for its security impact as it normally should have been.

By tracking and controlling all of the artifacts of the system development process, CM helps ensure that changes made to those artifacts cannot compromise the trustworthiness of the software as it evolves through each phase of the process. Coming from a systems development background, I have had the opportunity to practice CM hands on. Now that I am involved in security, I have found that good CM is no different than CM for security. Thus, practicing good CM is good security.



Requirements
- Security Risk Analysis
- Threat Modeling
- Security Requirements

Design and Build
- Secure Architecture and Design
- Security Minded Component Selection
- Vulnerability Analysis of Design
- Secure Coding

Test and Deploy
- Static Analysis of Source Code
- Security Testing

Operations and Maintenance
- Risk Analysis
- Vulnerability and Patch Management

Disposal
- Media Sanitization

Full Lifecycle
- Risk Management
- Configuration Management

Figure 1: *Phases of an SDLC*

Requirements

During requirements, a risk analysis of the initial functional requirements should be conducted. This initial analysis should form an important source of security requirements for the system and is the best way to start if the project team is having difficulty identifying initial security requirements. This helps address the familiar question *where do I start?*

Performing threat modeling and misuse/abuse cases are also important sources for developing appropriate security requirements as well as secure design. Threat modeling attempts to identify potential threats to the software, estimating the risk vulnerabilities may be exploited by these threats, and then defining countermeasures to mitigate the risks to the application. The development of misuse/abuse cases helps articulate scenarios in which security requirements can be derived. For example, a security requirement for validating user input could be developed to address a misuse/abuse scenario in which a user enters malicious scripting in the application's input field. Other sources for security requirements could be from policy, laws, regulations, standards, or best practices documents such as the following:

- Federal Information Security Management Act [4].
- Defense Information Systems Agency Security Technical Implementation Guides [5].
- National Institute of Standards and Technology (NIST) 800 series Special Publications [6].
- Comprehensive Lightweight Application Security Process [7].
- The Open Web Application Security Project (OWASP) best practices [8].

Additional security requirements discovered during the design, build, and test phases should be incorporated back into the system's requirements specification. Security flaws and defects found during testing should be analyzed to determine whether they originated with the system's requirements, design, or implementation and the root cause should be corrected in order to remove or mitigate the risk associated with that vulnerability. Tracing the origin of security defects is a form of measurement analysis that will help the organization identify where processes or products can be improved. This can tie into an organization's overall process improvement effort (e.g., Capability Maturity Model® Integration, International Organization for Standardization 9001, Six Sigma).

"Risk analysis should be repeated iteratively throughout the system's life cycle as different activities allow opportunities to identify new or changing risks."

Risk analysis can also help prioritize security requirements to focus resources on those components or functions that introduce the greatest vulnerabilities to the system. Specifically, security requirements that help mitigate the most critical risks identified from the risk assessment should be given priority over other requirements. In addition, it aids in striking a balance between security and functionality of the system.

Security requirements should not be treated separately from functional system requirements. Rather, the scope of the requirements development phase should be expanded to include security considerations. Ideally, both security and non-security requirements should exist in a solidified system requirements specification.

Design and Build

Integrating security into the selection or development of the architecture can be seamlessly achieved. When evaluating architectures for use, consider the security aspects of the candidate models or components, such as the following:

- Existing known vulnerabilities.
- Integration with other security products, such as an enterprise-level authentication product.
- Resiliency against certain attacks (e.g., cross-site scripting, structured query language [SQL] injection).[1]
- Ability to meet security requirements.

For example, when selecting commercial off-the-shelf products for integration, a search in the National Vulnerability Database (NVD) can determine if any known vulnerabilities exist for that particular product.

Comparing the access control mechanisms in Java frameworks such as *Struts* or *Spring* might be an example of a security consideration during architecture selection. Assessing how each framework handles security helps in developing the correct architecture – something that is often overlooked.

A secure architecture incorporates overlaps of security controls among components as well as built-in fall backs if the security controls of any one component are compromised. This is commonly referred to as *defense-in-depth*. For example, I typically advise customers to implement three layers of defense against SQL injection attacks. First, client-side input restrictions should be implemented for the purposes of thwarting casual attacks and providing faster response times in generating error messages. Then, input validation should be built into the application as a secondary line of defense. If the application receives repeated suspicious input at this point, it may be reasonable to assume that an attack is being attempted. Third, database calls should be constructed using parameterized queries to eliminate the possibility of SQL injection and to aid in the manageability of queries. In this architecture, there are fall backs in case a line of defense is compromised. Although all mechanisms provide an additional layer of defense, they also clearly have their own advantages in providing other features such as performance, incident detection, and manageability. Secure architecture of the application should also include countermeasures to compensate for vulnerabilities or inadequate assurance levels in its individual components and inter-component interfaces.

Designing for security involves both proactive detection and prevention of attacks, along with minimizing the impacts of successful attacks. Mechanisms such as fail-safe design, self-testing, exception handling, warnings to administrators or users, and self-reconfigurations should be designed into the application itself, while additional prevention, monitoring, and response mechanisms (e.g., application firewalls, intrusion detection systems, and security kernels) should be incorporated as defense-in-depth measures in the application's execution environment. The idea is that security mechanisms should be embedded into the application itself in addition to the traditional prevention, monitoring, and response mechanisms that are implemented around the perimeter of the application.

A fail-safe design, sometimes referred to as fail-secure, is a design that allows the application, in the event of an unrecoverable error, to fail without causing the application to be insecure. An example might be an application defaulting to no-access when a failed connection does not allow it to validate user permissions. At the source-code level, implementation of fail-safe may include having a default case in conditional code to protect the application in a situation where the other conditions are somehow not met. Self-testing is when the application can verify that its own security functionality is working properly. Robust exception handling is critical to security. It helps identify the source of problems and can be used for investigative purposes. Relying solely on exception handling by the Web server is not recommended, as it cannot capture specific actions taken on the application.

Designing for security involves the following key practices:

- Envision potential targets for attacks. What component of the system is most likely to be attacked?
- Analyze attacks from both external and internal sources; do not forget about malicious users inside the network.
- Design and include proven authentication methods, access policies, cryptographic algorithms, or other forms of security controls where appropriate.

Integration of security into the coding phase is relatively straightforward. For custom coding, developers should implement secure coding practices. For example, developers should ensure that user input is validated so that it only contains data that is expected, i.e., no malicious scripting or malformed input. Code should be thread-safe, so that during simultaneous execution by multiple threads, the code functions correctly and does not inadvertently access the data of other threads. Errors should be properly handled and debugging error messages should not be displayed to the user.

"Providing a set of secure coding standards does not guarantee those practices will always be implemented."

Security in implementation of purchased or acquired components focuses on implementing countermeasures and constraints to deal with known vulnerabilities in the individual components and their interfaces.

For custom-developed applications, coding standards are usually defined as part of the overall project effort. The project's coding standards should be augmented with extra standards for secure coding. For example, Sun has a set of security coding guidelines available [9] which would be effortless to integrate into the project's coding standards document. It cannot get much easier than this.

Providing a set of secure coding standards does not guarantee those practices will always be implemented. Therefore, validation of the source code is an essential activity. Much like peer review of source code, a security code review should be performed to assess the correctness and adequacy of the source code, but from a security standpoint. This technique is also referred to as *white-box testing*. The criteria selected to evaluate the source code should mirror the secure coding standards defined, or an agreed upon set of criteria based upon the level of risk an organization is willing to accept. Code analysis discovers subtle and elusive implementation errors before they reach testing or fielded system status. By correcting subtle errors in the code early, software development organizations can save engineering costs in testing and long-term maintenance [10]. Analysis can be performed manually or with automated tools. The Software Assurance Metrics And Tool Evaluation project at NIST attempts to classify software assurance tools and provides a comprehensive listing of code analysis tools [11].

Test and Deploy

Software security testing is not the same as traditional functional testing. The main objectives of software security testing are to identify vulnerabilities in the software and to ensure the secure behavior of software in the face of an attack.

Several techniques can be used for security testing, such as a vulnerability scan, penetration test, and security-oriented fault injection[2]. Vulnerability scans are performed with tools that attempt to detect application-level vulnerabilities (e.g., SQL injection, cross-site scripting) based on known attack patterns. Penetration testing attempts to break the application from the outside (the hacker's perspective). This can be accomplished manually by security specialists or in an automated fashion with tools such as brute-force attack tools.

What if a security code review has already been performed? Is it necessary to perform security testing since it is just another form of validation? Both validation techniques have their benefits and drawbacks. Security testing may simulate real-world attacks and exploitable vulnerabilities. Security testing may also be performed in conjunction with functional testing (assuming the capability is present). However, once vulnerabilities are found during testing, there is often limited time available to correct the problem. An ideal validation approach would be to complement source code analysis (performed during build) with security testing. With this approach, a minimal number of security flaws can be expected to come out of testing, which allows adequate time to correct them. Of course, a more pragmatic approach would be dependent on the amount of assurance required from the system and a definition of the overall security goal. An organization may choose to perform risk-based security code analysis or testing, assessing limited portions of the application based on risk.

Findings from security testing should also be fed back through the change control process as would non-security findings during functional testing.

Operations and Maintenance

Before the application goes into production, a security risk analysis should be performed to ensure that the application (new or updated) does not introduce an unacceptable level of risk. Results of security testing should be fed into the risk analysis, given there are viable threats to the vulnerabilities identified during testing.

Periodic risk assessments should be performed as the threat environment constantly changes. New attacks can uncover previously unseen vulnerabilities. It is also important to conduct risk analysis whenever major changes to the system occur. In the federal government, findings are fed into a plan of actions and milestones, where the mitigation is tracked to closure [12].

Major changes also require validation – a security code review and test. A good approach is to perform security code reviews and then attempt to exploit the more severe findings. This gives organizations a feel for the reality of their findings.

From an operational standpoint, it is important to constantly monitor security alerts and advisories that pertain to the technology implemented by the software since it is not uncommon for systems to fall victim to zero-day attacks[3]. A time-saving approach could be to subscribe to security alerts or feeds from product vendors or cyber-security sites such as the U.S. CERT or the NVD. This eliminates the need and dependency for administrators to constantly check Web sites for security updates. The counterpart to receiving advisories is an approach to address them once they are received. Therefore, it is critical to have a vulnerability and patch management program in place so that corrective action is taken, i.e., patches and fixes are applied in a consistent, timely manner.

Disposal

Security incidents are often seen when equipment is repurposed or disposed without completely eliminating records from hard drives or other data storage devices. Hardware and software should be appropriately sanitized, especially if the equipment will be re-used or repurposed.

Conclusion

Many of the security activities described in this article are simply an expansion of the current activities to include security considerations. Security is not as difficult to integrate into the SDLC as it may appear, and it is vastly more effective than bolting it on at the end. Integrated security ensures that the security mechanisms are adequate and effective, something that bolt-on security cannot boast.

Traditional approaches to security such as firewalls, intrusion detection systems, or server hardening are still important elements of security, but they are in no way the silver bullet for software security. They cannot protect the application itself against compromise. This is quite a paradox considering the application is the most visible of all the aforementioned components. The addition of these activities ensures that adequate security is baked-in to the end product and not sprinkled on, providing the system with resilience against attacks.◆

References

1. Gartner, Inc. "Recommendations for Security Administration, 2006." Qwest Communications 12 Dec. 2006 <www.mediaproducts.gartner.com/gc/webletter/qwest/issue4/gartner2.html>.
2. Carnegie Mellon University. CERT/CC <www.cert.org>
3. Havenstein, Heather. "Baked-In Security." ComputerWorld 21 Mar. 2005.
4. NIST. "Federal Information Security Management Act." <www.csrc.nist.gov/policies/FISMA-final.pdf>.
5. NIST. Security Configuration Checklists Repository. <www.checklists.nist.gov/repository/index.html>.
6. NIST Publications. Computer Security Resource Center <www.csrc.nist.gov/publications/nistpubs>.
7. Viega, John. Building Secure Software. Addison-Wesley Professional, 2001.
8. OWASP. The Open Web Application Security Project <www.owasp.org>.
9. Sun Developer Network. "Secure Coding Guidelines." 2000 <www.java.sun.com/security/seccodeguide.html>.
10. Lavenhar, Steven. "Code Analysis." BuildSecurityIn Portal. National Cyber Security Division. 28 Jan. 2006 <www.buildsecurityin.us-cert.gov/daisy/ bsi/articles/best-practices/code/214.html?branch=1&language=1>.
11. NIST. "Tool Taxonomy." <www.samate.nist.gov/index.php/Tool_Taxonomy>.
12. Daniels, Mitchell E. Jr. "Memoranda 02-01." 17 Oct. 2001 <www.whitehouse.gov/omb/memoranda/m02-01.html>.

Notes

1. SQL injection is a type of security exploit in which the attacker adds SQL code to a Web form input box to gain access to resources or make changes to data.
2. Fault injection is a testing technique where the application is fed anomalous input to reveal behavior.
3. A zero-day attack is an exploit against a vulnerability the same day the vulnerability becomes generally known.

Background

1. Goertzel, Karen Mercedes, et al, Security in the Software Lifecycle: Making Software Development Processes – and the Software Produced by Them – More Secure, Draft Version 1.2 (Aug. 2006), U.S. Department of Homeland Security.

About the Author

Kwok H. Cheng is an associate at Booz Allen Hamilton where he currently assists organizations in implementing security at the application level. He has a background in systems engineering, process improvement, and information assurance. Cheng has a master's degree in information and telecommunication systems, a certificate in information security management, and is a certified information systems security professional.

Booz Allen Hamilton
8283 Greensboro DR
McLean, VA 22102
Phone: (703) 902-3060
E-mail: cheng_kwok@bah.com

Cross-Domain Information Sharing in a Tactical Environment

Mel Crocker
General Dynamics Canada

Net-centric warfare in the full spectrum of operations mandates information sharing among non-traditional partners across security domains. This information sharing requires the exploitation of complex technologies and generates significant security challenges. Traditional information assurance solutions to support cross-domain information sharing have focused heavily on preventive measures, restricting information flow and reducing the risk of information compromise. This constraint on information flow directly opposes the duty of the warfighter to share information. A holistic solution involving robust software components, auditing, and permission management will reduce the risks of unauthorized information exposure to adequate levels without imposing severe information flow constraints.

Net-centric warfare is about *employing information age concepts to increase combat power in war and mission effectiveness in operations other than war* [1]. By linking sensor networks, command and control networks, and shooter networks, warfighters can achieve efficiencies in the full spectrum of operations by sharing information in a common operating environment. Unity of effort across organizational, national, technical and spatial boundaries is necessary. Warfighters have a *duty to share* information with others, and in the tactical environment it is not always obvious who needs the information and exactly how that information will be used. In some respects, sharing information is a leap of faith that the recipient will treat the information properly, not abusing the implied trust.

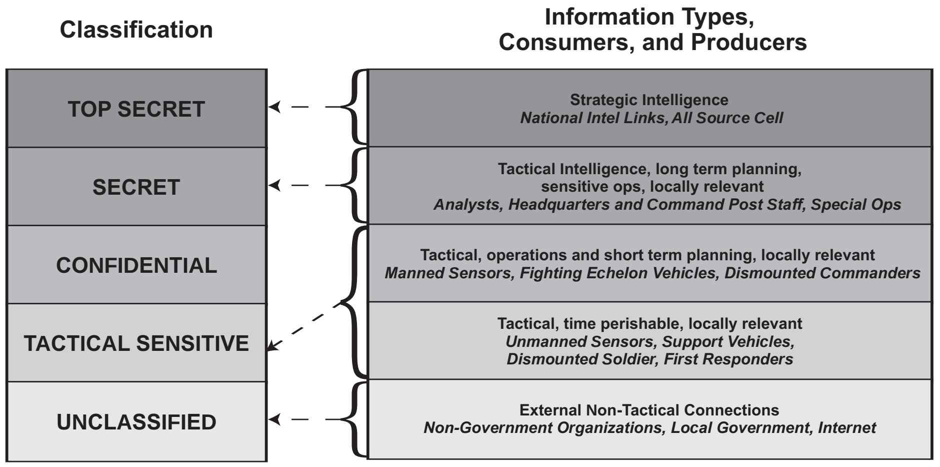
This article introduces aspects of the tactical environment and some of the complexities of sharing information in a tactical network, describes the security challenges and suggests a high-level security architecture that applies adequate measures without compromising the information sharing needs of the warfighter. Secure solutions to these types of complex net-centric problems are made achievable with the increased assurance that can be placed on well-developed and tested software.

The Tactical Information Environment

Information in modern tactical networks is generated from multiple sources: global positioning system receivers, unmanned and manned sensors, observations and recordings of individuals, higher command and intelligence networks, the Internet, and a variety of other sources. In modern operations, information must flow quickly from sensors to fusion processes to analysts and decision makers and, finally, to those who must execute action. Taking more than a few minutes from detection to action often significantly reduces the effectiveness of operations.

Beyond the need for quick and wide information flow, the tactical information environment is also made complex with differences in information sensitivity. As one moves from the fighting echelon of a tactical deployment back to national headquarters, there are fundamental differences in the sensitivity of data. For tactical elements in direct contact with the enemy, the majority of information processed is highly time perishable and generally focused on the following questions: Where am I? Where are my buddies? Where is the enemy, and what are his capabilities? Within a tactical headquarters environment, the information becomes more sensitive as plans are generated, intelligence is analyzed, and the larger tactical environment is monitored. Current government sensitivity labels and handling are based on definitions of sensitivities that were created for nationally sensitive information, only loosely relevant in the tactical environment. Executive Order 12958 describes three subjective classifications based on the damage resulting from compromise: TOP SECRET – *exceptionally grave* damage; SECRET – *serious* damage; and CONFIDENTIAL – damage to the national security [2]. These are relatively subjective groupings based on an interpreter's understanding of the anticipated impact of unauthorized disclosure and centered on national security. There is no time perishable consideration and all consideration is toward the affect on national security, not the impact on tactical operations; national security and tactical operations are related but are not the same thing. Figure 1 illustrates the five types of information and where they likely fit in current classifications. The figure introduces the term *tactical sensitive* for information falling between confidential and unclassified; it describes much of the information handled in a tactical environment.

Figure 1: *Information Sensitivity Classifications in the Tactical Environment*


Classification
Information Types, Consumers, and Producers
TOP SECRET
SECRET
CONFIDENTIAL
TACTICAL SENSITIVE
UNCLASSIFIED
Strategic Intelligence
National Intel Links, All Source Cell
Tactical Intelligence, long term planning, sensitive ops, locally relevant
Analysts, Headquarters and Command Post Staff, Special Ops
Tactical, operations and short term planning, locally relevant
Manned Sensors, Fighting Echelon Vehicles, Dismounted Commanders
Tactical, time perishable, locally relevant
Unmanned Sensors, Support Vehicles, Dismounted Soldier, First Responders
External Non-Tactical Connections
Non-Government Organizations, Local Government, Internet

Beyond the varied information sensitivities, warfighters also face the significant challenge of information overload and determination of correct distribution. To deal with this, Alberts and Hayes suggest that systems must transition from information *push* designs toward information *post* and *smart pull* designs.

Moving from a push to a post and smart pull approach shifts the problem from the owner of information having to identify a large number of potentially interested parties to the problem of having the individual who needs information identifying potential sources of that information. The second problem is a far more tractable one. This is because it is much easier for the individual who has a need for information to determine its utility than for the producer to make this judgment. [3]

This concept does not line up with the security tenet of limiting information distribution based on *need to know* and forces a new paradigm to providing adequate information assurance measures. The tactical environment will always have a need for some information to be pushed to consumers because there are alerts and critical developments that must be pushed to specific subscribers, but this *pushed* information is only a subset of the total information shared in the tactical environment.

When information of a like sensitivity is distributed within a defined community of interest, it is considered an information domain and is managed with a single security policy[1]. Often information must flow between domains; the cross-domain information exchanges must be sufficiently flexible to address the information sharing paradigm shift necessary for net-centric warfare.

Legacy Approaches to Cross-Domain Solutions (CDS)

Cross-domain information flow has always been considered a security-critical event and the risks associated with this type of transfer have been mitigated with a security guard. When the security guard is put into a system context, it is often referred to as a CDS. In the simplest sense, among other functions, a CDS confirms that information has been correctly downgraded when traveling from a higher domain to a lower domain, ensures that malware cannot move from the lower to the higher domain and confirms no information leakage from a higher to a lower domain. They are programmed for specific data formats, formally applying pre-established sets of rules and in general are very expensive. An example of a certified security guard is the software application Radiant Mercury which was developed under contract for the U.S. Navy by Lockheed Martin Corporation. When placed on a suitably trusted platform, this product automatically sanitizes, filters, and downgrades formatted classified documents.

Researchers from the Mitre Corporation have studied the evolution of guards toward better supporting the demands of today's warfighter and have concluded that guards should become more flexible, capable of handling information exchanges with libraries of approved schemas [4]. Mitre also identified critical functionality necessary in the security guard and some functionality that mandates a visibility beyond the guard to the connecting systems (e.g. workstation, server and user identification). Although some of these characteristics would be a large improvement over current guards, the suggested changes fail to address the information requirements of the modern tactical warfighter by constraining the timeliness, reach, and richness of information exchange. A *point* solution such as a security guard cannot effectively satisfy the information sharing demands of the tactical warfighter and a holistic system solution is required.

Beyond that, legacy security guards do little to mitigate the risks posed by insiders. A level of trust is placed in the individual charged with assigning a sensitivity label to information and in those who handle or consume the information. With the asymmetric threat posed by an insider, it becomes even more important to ensure individual trust is not abused[2], and if it is abused, to detect this transgression quickly, prevent further damage and provide an adequate forensic trail to hold the attacker accountable. Previous CDS approaches imposed preventive measures on the unauthorized disclosure of information instead of focusing on the trust placed in the individual. Information that was incorrectly marked and/or carefully prepared to avoid rules was unlikely to be detected by a CDS. Although individuals will always be subject to compromising trust relationships, these transgressions would be more detectable in a system solution.

Security Solution for Tactical Cross-Domain Information Sharing

The tactical environment demands a security solution that provides measures to keep the information protected, and that allows for timely, widely distributed and rich information exchanges. Solutions today must reduce the risks associated with information compromise such that they are significantly outweighed by the benefits of the system, thereby providing commanders with the means to exercise their duty to share information.

A number of significant factors have recently changed, creating the opportunity for new approaches to cross-domain information sharing.

Certification Advances

The suggested architecture has to be certified by appropriate authorities and accredited by commanders for operation in specific environments. Unfortunately, certification and accreditation (C&A) have become significant challenges for systems, leading to solutions such as the security guards discussed earlier. *Instead of being viewed as helpful, C&A is considered a hindrance. It is neither timely nor cost-efficient in an era when technology advances are coming faster than ever* [5]. The Defense Information Assurance Certification and Accreditation Process (DIACAP), still in draft format[3], introduces changes that provide a framework for certifying system solutions … *to support the paradigm shift from* need to know *to* need to share [5]. The DIACAP is applicable to tactical information sharing and introduces a process that could be used to certify and accredit the high-level security solution proposed in this article.

Technology Advances

Several technologies are creating opportunities for better cross-domain security solutions.

1. The Trusted Computing Exemplar (TCX) project is creating a framework for rapid high assurance system development, addressing how high assurance software components can be built [6]. With the system solution envisaged in this article, several high assurance components will be required at various places in the system and the TCX project identifies a process prescribing how these types of components can be built. Moreover, there are a number of companies who have significantly matured their software development processes, achieving the Software Engineering Institute's Capability Maturity Model and Capability Maturity Model Integration Level 5. Beyond mature software development processes, the improvements in verification of software have also been significant and are becoming the focus of intense research [7]. Creating software that predictably and verifiably does what it purports to do and nothing more is becoming achievable within reasonable expense. All these elements are critical to building a system solution.

2. The Advanced Encryption Standard (AES) was approved in Federal Information Processing Standards Publication 197 dated 26 Nov. 2001 to encrypt unclassified U.S. government traffic. In June 2003, the National Security Agency (NSA) approved AES to protect classified U.S. traffic, an unprecedented action in the world of high-assurance encryption [8]. Because the algorithm is publicly available, coalition partners can independently implement the algorithm and with a common key, they can securely exchange information.
3. The Trusted Computing Group (TCG)[4], an alliance of manufacturers, is in the process of establishing a number of relevant security hardware and system standards, effectively creating a framework for secure system solutions. The TCG recognizes the critical link with hardware, and several manufacturers are beginning to market compliant equipment. Regarding the solution suggested in this article, TCG compliant equipment would create an affordable, stable hardware base for the high assurance software components.
4. Persistent information storage is becoming very inexpensive and readily available devices can store immense quantities of information. This is important to support archiving audit logs. Protecting the integrity of and controlling access to the audit logs can be securely accomplished using the measures identified in the Trusted Platform Module specifications. Draft *NIST Special Publication 800-86, Guide to Computer and Network Data Analysis: Applying Forensic Techniques to Incident Response*, and draft *Special Publication 800-92, Guide to Security Log Management* provide considerable rigor to audit processes and log management.
5. The Group Security Association Key Management Protocol (GSAKMP) *provides a security framework for creating and managing cryptographic groups on a network* [9]. If information can be tagged for a particular community of interest, access to that information can be managed by cryptographic mechanisms. Warfighters only need to send the information to a community of interest, and the framework provided by GSAKMP will ensure secure access is managed for participants. Separating the complexity of correctly tagging information from distribution decisions will allow for a more efficient information sharing environment.
6. There have been a number of significant advances recently toward certified components leading toward a certified Multiple Independent Levels of Security (MILS) architecture [10]. A MILS architecture leads toward a degree of confidence in the separation of information within the system, avoiding so much technical complexity that the system cannot practically be built. This creates well-enforced system sandboxes where software can be forced to execute only within approved parameters. The High Assurance Platform (HAP) is a computer that provides MILS capabilities using industry standard commercial hardware, software and applications, and should be available to a narrow community in 2007. It is intended to provide NSA certified separation to multiple operating systems running simultaneously in different security domains[5].
7. The Department of Defense (DoD) Discovery Metadata Specification was created to allow for efficient information discovery in US government networks [11]. It includes a number of tags that are relevant to security, including classification, declassification date, dissemination controls, and others. Moreover, search engines have become increasingly more efficient with hybrid designs of crawler/spider based components and human powered directories. Despite the growing quantity of information available, finding relevant information is becoming easier due to the use of metadata labels and strong search engines.

A systemic information assurance approach must provide layered security measures that reduce the need for information content filtering measures. The solution relies on increasing the strength and reliability of accountability, detective and reactive measures, and consequently increases flexibility in information sharing. The following high-level solution builds on the advantages presented by recent maturations in select technologies.

Encryption to support confidentiality and integrity of information flows should be established from source to destination, so devices at the boundary will not be able to examine the contents of packets. This means that some of the traditional functionality of the CDS must be pushed to the information sources, in most cases data terminals. It is important to note that the destination need not necessarily be another data terminal but could be traffic destined for a community of interest.

The following functionality must exist at information source points, often personal computers:

- Trusted identification and access control measures must be resident in the source data terminals. These measures link user triggered actions to individuals and confirm privileges before allowing actions. Systems and protocols provide the means to manage identities across disparate networks with a high degree of confidence and minimum inconvenience to the user community. Regarding authorization, the use of X.509 based attribute certificates and a Privilege Management Infrastructure offers considerable flexibility to handle role based authority [12] and progress has been made extending Public Key Infrastructures into tactical environments.
- Trusted audit measures must be resident on the data terminals to capture all security relevant events. With the establishment of the TCG standards and resulting hardware, the audit logs can be securely protected and with the availability of inexpensive storage, the logs can hold a tremendous amount of information before needing to be rolled over.
- Trusted domain separation must exist on the data terminals. There is considerable research into making trusted operating systems more accessible and commercial operating systems more secure, providing sufficient flexibility to strike the right risk exposure and functionality. Moreover, with the establishment of TCG standards and hardware, the increased confidence in the operating systems and software will be strongly based on trusted hardware. This should make domain separation on desktops achievable and affordable in the near term.
- Trusted encryption measures with an appropriate algorithm must provide adequate confidentiality and integrity protection for information flows between data terminals. Trusted Network Connect from the TCG offers an assured encryption solution and the digital signature, random number generation and protected storage of the Trusted Platform Module, again from the TCG, offers the other necessary primitives for a secure solution.
- Malicious content filtering that detects and prevents the execution of

malware must exist on all data terminals. With host firewalls and advances in malware protection products, networked computers can connect to the Internet with a degree of confidence. In a tactical environment, the risk of attack is far reduced and the potential for successful malware introduction is also reduced.

- There must be trusted, locally controlled interfaces that allow for movement of select data from one domain on a data terminal to another domain on the same terminal. The term *controlled interface* is not carefully defined in this article, but in general, it mediates information exchanges, looking for defined transgressions when moving data between domains. Effectively a controlled interface gives authorized users some flexibility to move data electronically between domains.

Within the connecting network, there must be a number of complimentary services:

- An identity and permission management service must provide complimentary functionality to the capabilities needed at the information sources.
- Network and distributed intrusion detection and reaction services must exist. Other than receiving alerts from network intrusion detection sensors, this service would also receive alerts from host based intrusion detection sensors, collect log data for detailed analysis and react to events based on policy.
- A policy management service provides the needed flexibility for a tactical network. This service would establish policies that relate to the network configuration and security posture. These policies would be dynamic based on changes in information flow requirements, changing threats and various other influences.
- A service that provides security association and key management is needed to support GSAKMP.

A boundary protection system should contain the following functionalities at the network boundaries.

- Identity and access control to ensure the users passing information or drawing information across the domain boundary are authorized to do so.
- Flow control measures that can accommodate the need for supporting quality of service information exchanges such as near real-time graphical collaboration sessions, and control network ingress from unwanted sources and/or unwanted traffic.
- Audit measures that work with intrusion detection and prevention systems to detect malicious activity and/or exposures in a timely fashion and provide non-repudiation of information flows.
- The configuration of boundary protection services must be dynamically configurable by policy, allowing information flow based on tactical conditions.

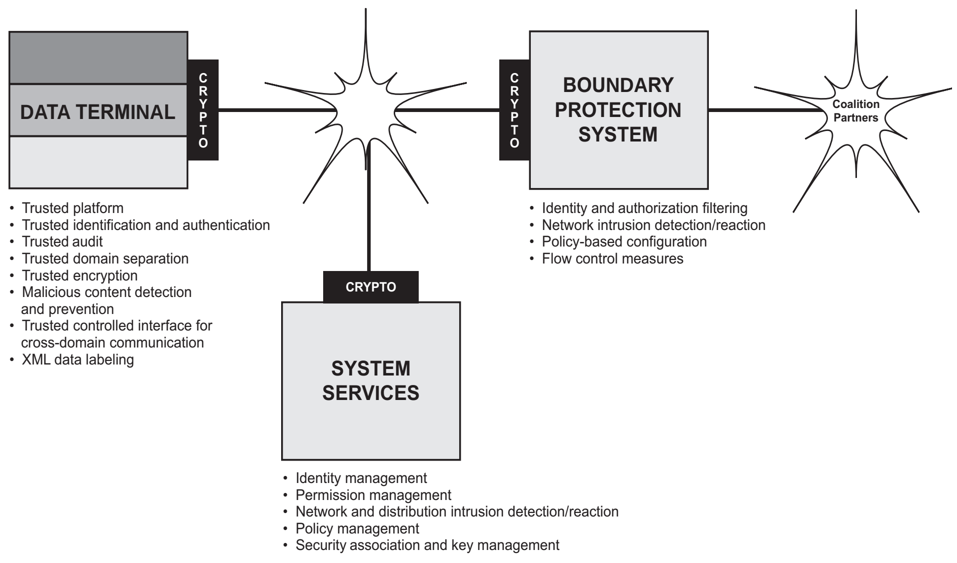

DATA TERMINAL
CRYPTO
CRYPTO
BOUNDARY PROTECTION SYSTEM
Coalition Partners
• Trusted platform
• Trusted identification and authentication
• Trusted audit
• Trusted domain separation
• Trusted encryption
• Malicious content detection and prevention
• Trusted controlled interface for cross-domain communication
• XML data labeling
• Identity and authorization filtering
• Network intrusion detection/reaction
• Policy-based configuration
• Flow control measures
CRYPTO
SYSTEM SERVICES
• Identity management
• Permission management
• Network and distribution intrusion detection/reaction
• Policy management
• Security association and key management

Figure 2: *Suggested High-Level Solution*

This high-level architecture (Figure 2) has a number of implementation challenges that require deeper analysis, but these have not been identified or explored in this article for brevity purposes.

Conclusion

To support the unity of effort necessary in today's combat environment, warfighters have a duty to share information widely and quickly in rich exchanges, some of which must cross security domains. This article suggests a holistic high-level solution to securing cross-domain exchanges that will not excessively constrain the exchanges, taking advantage of advances in technology and policy. The solution effectively takes some of the trust and functionality originally resident in traditional CDS and moves it into information sources, system services, and boundary protection devices.

Although the solution suggested here has been applied to the tactical environment, elements of the system solution may lend itself to other environments with similar problem spaces. Instead of tactical domains, one could consider the domains relevant in medical information systems. Patients must securely share private information with family general practitioners, and occasionally general practitioners must share elements of this information with specialists. The exchange between patient, general practitioner, and specialist creates a small community of interest. At the same time, some of this information may be useful to those needing statistics, but the posting agency may not really be aware of the information needs of the authorized consumers and may not be best able to manage the makeup of the authorized consumers. Managing access might be better placed with others whose primary expertise is privacy, access control, and information presentation. Throughout these exchanges, actions must be logged to ensure violations can be handled quickly.

This article has not proposed any dramatic new technologies; it has simply suggested re-positioning some relatively well-understood security functionality to non-traditional places in the network in the hopes of satisfying the information sharing needs of the warfighter.◆

References

1. DoD. "Network Centric Warfare." DoD Report. 27 July 2001 <www.dod.mil/nii/NCW/>.
2. Bush, George W. Further Amendment to Executive Order 12958 <www.whitehouse.gov/news/releases/2003/03/20030325-11.html>.
3. Alberts, D.S., and R.E. Hayes. "Power to the Edge: Command and Control in the Information Age." CCRP (June 2003): 14-15.
4. Reed, Nancy. "Security Guards for the Future Web." Technical Report 04W0000092. Mitre Corporation,

2004 <www.mitre.org/work/tech_papers/tech_papers_05/05_0166/05_0166.pdf>.
5. Wierum, Jenifer M. "Defense Information Assurance Certification and Accreditation Process and the Global Information Grid Information Assurance Architecture." 10th International Command and Control Research and Technology Symposium The Future of C2, Mar. 2005.
6. Irvine, Cynthia E., Timothy E. Levin, Thuy D. Nguyen, and George W. Dinolt. "The Trusted Computing Exemplar Project." Proc. of the 5th IEEE Systems, Man and Cybernetics Information Assurance Workshop, United States Military Academy, West Point, NY, 10-11 June 2004.
7. Jones, Cliff, Peter O'Hearn, and Jim Woodcock. "Verified Software: A Grand Challenge." IEEE Computer 39.4 (2006).
8. Committee on National Security Systems. "National Policy on the Use of the Advanced Encryption Standard to Protect National Security Systems and National Security Information." U.S. CNSS Policy No. 15 Sheet No 1. June 2003.
9. Internet Engineering Task Force. "Group Secure Association Group Management Protocol." Internet Draft. 2005 <www3.ietf.org/proceedings/06mar/IDs/draft-ietf-msec-gsakmp-sec-10.txt>.
10. Alves-Foss, Jim, Carol Taylor, and Paul Oman. "A Multi-layered Approach to Security in High Assurance Systems." Proc. of the 37th Hawaii International Conference on System Sciences, 2004.
11. Magar, A. "Investigation of Technologies and Techniques for Labeling Information Objects to Support Access Management." Defense Research and Development Canada, Report DRDC Ottawa CR 2005-166. 2005
12. Chadwick, David. "The X.509 Privilege Management Infrastructure." Proc. of the North Atlantic Treaty Organization Advanced Networking Workshop on Advanced Security Technologies in Networking, Bled, Slovenia, June 2003. University of Salford, 2003 <http://sec.isi.salford.ac.uk/Papers.htm>

Notes

1. A security domain is defined by the Internet Security Glossary as *an environment or context that is defined by a security policy, security model, or security architecture to include a set of system resources and the set of system entities that have the right to access the resources* <www.ietf.org/rfc/rfc2828.txt>.
2. The John Anthony Walker Jr. story of insider espionage activities over eighteen years can be found at <www.crimelibrary.com/terrorists_spies/spies/walker/1.html>.
3. A draft version of the DIACAP is available at <http://iase.disa.mil/ditscap/ditscap-to-diacap.html#diacap>.
4. More information on the TCG and its standards can be found at <www.trustedcomputinggroup.org>.
5. The HAP is described at <www.gdc4s.com/contacts/user_conf/topics.cfm>.

About the Author

Mel Crocker is the information assurance technical lead for the Land Command Support System program within General Dynamics Canada. He has a master's degree in software engineering from Royal Military College, and a Bachelor of Science in math and physics.

General Dynamics Canada
1020-68th AVE N.E.
Calgary, AB T2E 8P2
Phone: (403) 295-5075
E-mail: melvin.crocker@gdcanada.com

Project Management Using Random Events

Are you tired of having to constantly report problems to your boss? Are you tired of coming up with yet another lame excuse for problems caused by others? Let CrossTalk save you some time with the Department of Defense General-Purpose Excuse and Explanation Memo form (DODGE EM). Simply get yourself a single six-sided die, start rolling, and fill in the blanks below. This gives you well over 45,000 possible explanations from which to choose. You will save so much time with paperwork, you might actually be the first developer on your project to actually spend more time developing than doing paperwork!

TO: --Roll Dice, Insert from **Table 1**--

I am truly sorry to report that my project is --Roll Dice, Insert from **Table 2**--.

I first realized this --Roll Dice, Insert from **Table 3**--.

I'm sorry, but this happened because we --Roll Dice, Insert from **Table 4**--.

To put us on the right path, we need to --Roll Dice, Insert from **Table 5**--.

It would really help us if you would --Roll Dice, Insert from **Table 6**--.

This won't happen again – because next time, I will --Roll Dice, Insert from **Table 7**--.

Sincerely,
--Insert your name here--

Table 1
1. My boss.
2. My boss' boss.
3. The General Accounting Office.
4. Congressional Oversight Committee.
5. My mommy.
6. Those who read this after I go.

Table 2
1. Running over schedule.
2. Running out of money.
3. REALLY running out of time and money.
4. Making the developers nauseous.
5. Forcing those associated with the project to rewrite their resume.
6. Being referred to in-house as the "resume stain."

Table 3
1. This morning.
2. In the middle of the night, after having the $2.99 "all-you-can-eat" burrito special.
3. When faced with certain and imminent discovery.
4. After everybody else figured it out first months ago.
5. In spite of massive amounts of anti-depressants and competent psychiatric care.
6. After all of the developers took up meditation and fled the country.

Table 4
1. Ignored the advice of all the consultants.
2. Listened to every single consultant.
3. Re-scoped the requirements 14 times.
4. Never had any requirements.
5. Spent all the budget, and still never had any requirements.
6. Built it and made it to test without any real requirements.

Table 5
1. Hire a consultant who knows something about security.
2. Fire the consultants who know nothing about security.
3. Switch to Ada – a secure language.
4. Realize that 53 designers and no coders is a bad mix.
5. Figure out who the users might be and try talking to one about security.
6. Have the developers take up interpretative ballet as a hobby.

Table 6
1. Show up occasionally to the stakeholder meetings.
2. Quit showing up to the stakeholder meetings.
3. Micromanage somebody else for JUST a few days.
4. Show up late, leave early, and stay behind closed doors.
5. Realize that your degree in sanitation engineering does not make you a technical expert – and that security DOES NOT mean wearing a Depends.
6. Attend church regularly and pray for the project – it's our only hope.

Table 7
1. Read CrossTalk regularly.
2. Read CrossTalk regularly.
3. Read CrossTalk regularly.
4. Read CrossTalk regularly.
5. Read CrossTalk regularly.
6. Read CrossTalk regularly.

— David A. Cook, Ph.D.
Senior Research Scientist and Principle Member of the Technical Staff
The AEgis Technologies Group, Inc.
dcook@aegistg.com

Homeland Security

Software Assurance Program

Software is essential to enabling the nation's critical infrastructure. To ensure the integrity of that infrastructure, the software that controls and operates it must be reliable and secure.

Security must be "built in" and supported throughout the lifecycle.

Visit http://BuildSecurityIn.us-cert.gov to learn more about the practices for developing and delivering software to provide the requisite assurance.

Sign up to become a free subscriber and receive notices of updates.

http://BuildSecurityIn.us-cert.gov

The Department of Homeland Security provides the public-private framework for shifting the paradigm from "patch management" to "software assurance."

CrossTalk / 517 SMXS/MXDEA
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

PRSRT STD
U.S. POSTAGE PAID
Albuquerque, NM
Permit 737

CrossTalk is co-sponsored by the following organizations:

NAVAIR

Homeland Security